Cost-Effective App User Allocation in an Edge Computing Environment

Phu Lai, Qiang He, John Grundy, Feifei Chen, Mohamed Abdelrazek, John Hosking, and Yun Yang

Abstract—Edge computing is a new distributed computing paradigm extending the cloud computing paradigm, offering much lower end-to-end latency, as real-time, latency-sensitive applications can now be deployed on edge servers that are much closer to end-users than distant cloud servers. In edge computing, edge user allocation (EUA) is a critical problem for any app vendors, who need to determine which edge servers will serve which users. This is to satisfy application-specific optimization objectives, e.g., maximizing users' overall quality of experience, minimizing system costs, and so on. In this paper, we focus on the cost-effectiveness of user allocation solutions with two optimization objectives. The primary one is to maximize the number of users allocated to edge servers. The secondary one is to minimize the number of required edge servers, which subsequently reduces the operating costs for app vendors. We first model this problem as a bin packing problem and introduce an approach for finding optimal solutions. However, finding optimal solutions to the $\mathcal{NP}$-hard EUA problem in large-scale scenarios is intractable. Thus, we propose a heuristic to efficiently find sub-optimal solutions to large-scale EUA problems. Extensive experiments conducted on real-world data demonstrate that our heuristic can solve the EUA problem effectively and efficiently, outperforming the state-of-the-art and baseline approaches.

Index Terms—Edge computing, fog computing, user allocation, optimization, resource allocation

1 Introduction

Mobile and Internet-of-Things (IoT) devices, including smartphones, wearables, sensors, etc., have become extremely popular in modern society [2]. The rapid growth and advances of mobile and IoT devices have fueled the variety and sophistication of mobile and IoT software applications such as natural language processing, facial recognition [3], interactive gaming [4], etc. Those applications usually require intensive processing power and high energy consumption. Due to the limited computing capabilities and battery power of mobile and IoT devices, heavy computation tasks are often offloaded to the app providers' servers in the cloud to be processed. However, this usually comes with high network latency and unpredictable delays, due to the long distance between the cloud servers and end-users, as well as a performance bottleneck that is often caused by the centralized network architecture (e.g., sudden surges of users or heavy workloads, network issues, and so on). In addition, as the number of connected devices is rapidly increasing, predicted to be around 32 billion by 2023 [2], plus the continuously increasing network traffic and computational workload, online app vendors are facing the challenge of maintaining a reliable and low-latency connection to their users, which is one of the key quality-of-service (QoS) requirements [5].

This is an extended and revised version of a preliminary conference paper that was presented in ICSOC 2018 [1].

- *P. Lai, Q. He, and Y. Yang are with the School of Software and Electrical Engineering, Swinburne University of Technology, 3122, Australia. E-mail: tlai, qhe, yyang@swin.edu.au.*
- *J. Grundy is with the School of Information Technology, Monash University, 3168, Australia. E-mail: john.grundy@monash.edu.*
- *F. Chen and M. Abdelrazek are with the School of Information Technology, Deakin University, 3125, Australia. E-mail: mohamed.abdelrazek, feifei.chen@deakin.edu.au.*
- *J. Hosking is with the School of Science, University of Auckland, Auckland, New Zealand. E-mail: j.hosking@auckland.ac.nz.*

To tackle this issue, Cisco [6] has proposed the fog computing paradigm – sometimes called *edge computing* – in which computing, networking, and storage resources are distributed closer to the edge of the network by a number of intermediate *edge servers* deployed in close proximity to end-devices, or end-users. This paradigm offers lower network latency than the conventional centralized cloud computing paradigm. It also offers great scalability by enabling computation and storage at the edge of the network. This significantly reduces the data transmission (and bandwidth consumption) between end-users and the cloud [7]. This is particularly important for high volume streaming applications or critical systems such as autonomous traffic systems, health care, or online gaming, which require real-time decision making. In edge computing, app vendors hire existing edge servers to host their applications serving their users. Thin clients – such as wearables, sensors, or smartphones – all that have limited battery power, storage, and computing capabilities, benefit from this architecture by the ability to offload intensive computation tasks to the distributed edge servers near them [8]. In this way, the central cloud is not required to perform all the computation tasks single-handedly, which is highly resource-demanding, has a higher chance of being a performance bottleneck, and generally incurs higher network latency to users.

In a mobile edge computing (MEC) environment, a number of edge servers are deployed by edge infrastructure providers like AT&T or Vodafone in a distributed fashion (usually near cellular base stations [9]) so that they can cover different geographical areas. Users within an edge server's coverage can connect to the edge server via LTE, 4G, or 5G [10]. Edge infrastructure providers are responsible for ensuring a low latency connection to users. A user's latency requirement is assumed to be satisfied as long as the user is allocated to an edge server. The coverage areas

of adjacent edge servers usually partially overlap to avoid non-service areas – areas that are not covered by any edge server [1], [11]. A user located in the overlapping area can connect to one of the edge servers covering them (*proximity constraint*) that has sufficient computing resources (*capacity constraint*) such as CPU, storage, bandwidth, or memory. Note that from the app vendor's perspective, they need to ensure that the resources hired on an edge server will suffice to accommodate their users allocated to that edge server during the allocation process. How much of the hired resources will be used is dependent on the users at runtime. Compared to a cloud data center, a typical edge server has very limited computing resources [12], [13], which leads to the need for an effective resource allocation strategy.

Edge servers' capacities, current workloads, coverage, proximity to users, and the number of users to allocate can be obtained or calculated at runtime. Due to the limited resources of the edge servers, an edge server might not be able to serve all the users within its coverage. Since some of the users might also be located in other edge servers' coverage, an app vendor can allocate them to those edge servers to share the workload with the overloaded servers. Due to the aforementioned constraints, there might be some users that cannot be assigned to any edge server. Those users will be connected directly to a remote cloud server, which is not desirable. Therefore, the optimization objective here is to maximize the number of users allocated to edge servers, which ensures the QoS from the app vendor's perspective. In the meantime, the number of edge servers required to serve those users (or the number of active edge servers) needs to be minimized. This will ensure the cost-effectiveness of the allocation by cutting down the app vendor's costs of hiring edge servers to serve their app users [**?**], [1] and improving the resource utilization on edge servers [**?**], [14]. In fact, minimizing the number of required servers is one of the key objectives in the server consolidation problem in cloud computing [15], [16]. In this paper, we study quasi-static scenarios where users are relatively static, not roaming across edge servers quickly [**?**], [4], [11], [17], e.g., surveillance cameras, traffic sensors, mobile, or IoT users who stay in one location.

The above problem is referred to as an *edge user allocation* (EUA) problem [**?**], [1], [18], [19]. This problem has been modeled as a bin packing problem and proven to be $\mathcal{NP}$-hard in our previous work [1]. As demonstrated in [1], the EUA problem is extremely computationally expensive to solve optimally in a large-scale scenario due to the dense distribution and limited computing resources of edge servers. It takes up to 23 seconds to find an optimal solution when there are just around 512 users and 125 edge servers, which is unacceptable for applications or services that require real-time or near real-time decision making. Therefore, we introduce Most Capacity First (MCF), a heuristic approach for efficiently finding a sub-optimal solution to the EUA problem. Note that the EUA problem has a number of variants with different assumptions and objectives, depending on specific applications and services. For example, [18] aims to maximize the number of allocated users and minimize the number of user reallocations when dealing with user mobility; [**?**] aims to maximize the number of allocated users with minimum system costs; and [19] aims to maximize users' overall quality of experience (QoE). The main contributions of this paper include:

- In our previous work [1], we formally model the EUA problem as a variable-sized vector bin packing (VSVBP) problem, which is $\mathcal{NP}$-hard. We then optimally solve this problem using the Lexicographic Goal Programming technique.
- Due to the $\mathcal{NP}$-hardness of the problem, finding an optimal solution is intractable in a scenario that involves a great number of users and edge servers. To effectively deal with the high complexity, this paper proposes MCF – a heuristic approach for finding a sub-optimal EUA solution efficiently.
- Extensive evaluations are conducted on a real-world dataset to demonstrate the effectiveness and efficiency of the proposed approaches. The results show that our approaches outperform the state-of-the-art and baseline approaches.

The remainder of the paper is organized as follows. Section 2 motivates this research with an example. Section 3 introduces the VSVBP problem, based on which we formulate the EUA problem in Section 4. Section 5 presents our optimal and heuristic approaches for solving the EUA problem, which are evaluated in Section 6. Section 7 reviews the related work and Section 8 concludes this paper.

2 Motivating Example

A representative application that can leverage the low latency provided by edge computing is large-scale cloud gaming [20] - the fastest growing gaming model [21]. This model has made many online game platforms, such as Hatch[1] and Sony PlayStation Now[2], more accessible to thin-client mobile players since the resource-expensive game application instances are running on powerful servers in the remote cloud. Let us consider an increasingly popular virtual reality game, which requires a great amount of computing power for graphics rendering. The centralized cloud model allows mobile devices to offload heavy computation tasks to the servers in the cloud. However, this approach often introduces significant network delays because of the long distance between the players and the remote cloud servers. Therefore, pushing computing power closer to players with edge computing is a promising solution to this problem.

Fig. 1 shows a small example of an MEC environment with 8 players $u_1, ..., u_8$ and 3 edge servers, $s_1, ..., s_3$. Each edge server covers a particular geographical area and has a specific amount of computing resources (CPU, RAM, storage, and bandwidth). An edge server can only serve players within its coverage (*proximity constraints*). For example, player u_1 can be allocated to either edge server s_1 or s_3 only. Edge server s_2 cannot serve player u_1 since player u_1 is outside its coverage area. In an edge computing environment, the computing resources of an edge server, such as CPU, RAM, bandwidth, etc., are usually shared by multiple users. Thus, in addition to the proximity constraints, the game vendor needs to take

1. www.hatch.live/
2. www.playstation.com/en-gb/explore/playstation-now/

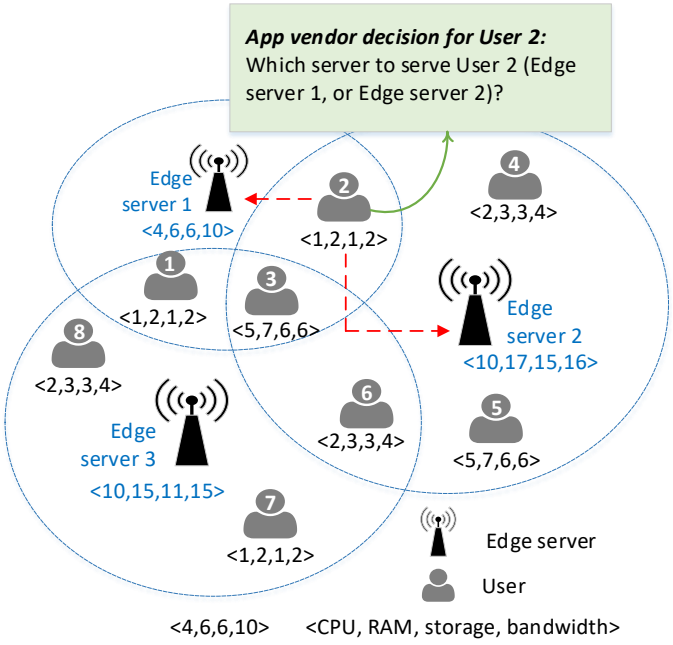


Fig. 1: Example edge computing environment

into account various *capacity constraints*, i.e., whether the available computing resources on an edge server suffice to serve the players allocated to it. In Fig. 1, each edge server has a limited amount of computing resources, denoted as a vector $\langle CPU, RAM, storage, bandwidth \rangle$. Different players might require different amounts of computing resources. For instance, edge servers that serve players who select higher graphics settings (4K graphics quality, spatial anti-aliasing, etc.) consume more resources to facilitate high-quality graphics rendering. The total resource requirements of players allocated to an edge server must not exceed the available computing resources on that edge server. There are five players located within the coverage of edge server s_3, whose aggregated resource requirement is $\langle 11, 17, 14, 16 \rangle$, which exceeds the available resources of server s_3 ($\langle 10, 15, 11, 15 \rangle$). Thus, the game vendor needs to allocate some of those players to other edge servers. The objective is to allocate as many players to as few edge servers as possible.

A possible solution to the EUA problem in Fig. 1 is to allocate player u_1 to server s_1, players u_2, u_3, u_4, and u_6 to server s_2, and players u_7, and u_8 to server s_3. Player u_4 cannot be allocated to server s_2 since the server is already serving other players and no longer has sufficient resources to serve player u_4. Neither proximity nor resource constraints are violated in this way. However, this is not the optimal solution in terms of QoS or cost-effectiveness since only seven out of eight players are allocated to edge servers, and all three edge servers are required. A better solution would be to allocate players u_2, u_4, u_5, and u_6 to server s_2, and players u_1, u_3, u_7, and u_8 to server s_3. This allocation allows all eight users to be served by edge servers; furthermore, it does not require edge server s_1 at all.

Finding an optimal solution to the EUA problem is not trivial, especially in a large-scale scenario with numerous users covered by numerous edge servers. Fig. 1 is a very small-scale example. In a real-world EUA scenario, there would be many more players (or app users in general) and edge servers, resulting in a much larger solution space for the EUA problem. Assuming that there are n users and m edge servers, the solution space may consist of up to m^n possible solutions. Thus, an effective and efficient user allocation approach is needed for app vendors in such scenarios.

3 Background

Definition 1. Classic Bin Packing (BP) Problem. *Given an infinite supply of identical bins $\mathcal{S} = \{s_1, s_2, ...\}$ with capacity C and a set of n items $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. Let a value w_j be the size of item u_j that satisfies $0 < w_j \leq C, \forall u_j \in \mathcal{U}$. The objective is to pack all the given items into the fewest bins possible such that the total item size in each bin must not exceed the bin capacity C: $\sum_{u_j \in \mathcal{U}_{s_i}} w_j \leq C, \forall s_i \in S$, where $\mathcal{U}_{s_i}$ is the set of items placed in bin s_i.*

The only constraint in the classic BP problem is that the total size of the items packed into a bin must not exceed the capacity of the bin. This combinatorial optimization problem is known to be an $\mathcal{NP}$-hard problem [22].

Definition 2. Variable-Sized Bin Packing (VSBP) Problem. *Given a limited collection of k bin sizes such that $1 = size(B^1) > size(B^2) > ... > size(B^k) > 0$, there is an infinite supply of bins for each bin type B^l, where $l = 1, ..., k$. Let $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ be the list of bins needed for packing all items. Given a list of n items $\mathcal{U} = \{u_1, u_2, ..., u_n\}$, each with item size $w_j \in (0, 1]$, where $j = 1, ..., n$, the aim of the VSBP problem is to pack all items into bins so that the total size of items in each bin must not exceed the bin capacity, and the total size of the required bins $\sum_{s_i \in \mathcal{S}} size(s_i)$ is the minimum.*

In the classic BP problem, all bins are homogeneous with the same bin capacity. VSBP is a more general variant of the classic BP in which a limited collection of bin sizes is allowed. VSBP aims at minimizing the total size of the bins used, which is different compared to the aim of the classic BP problem.

Definition 3. Vector Bin Packing (VBP) Problem. *Given a set of n items $\mathcal{U} = \{u_1, u_2, ..., u_n\}$, the size of item $u_j \in \mathcal{U}$ is denoted as d-dimensional vector $w_j = \langle w_j^1, w_j^2, ..., w_j^d \rangle$, where $w_j \in [0, 1]^d$. One is given an infinite supply of identical bins $S = \{s_1, s_2, ...\}$ with capacity $C = \langle 1^1, 1^2, ..., 1^d \rangle$. The aim is to pack the set $\mathcal{U}$ into a minimum number of bins $s_i \in \mathcal{S}$ such that $\|\sum_{u_j \in \mathcal{U}_{s_i}} w_j\|_\infty \leq 1, \forall s_i \in \mathcal{S}$, where $\mathcal{U}_{s_i}$ is the set of all items packed in bin s_i.*

In the classic BP problem, the size of an item is presented as a single aggregation measure. In contrast, the size of an item in the VBP problem is associated with a multi-dimensional vector. The aim remains similar, in which the sum of packed item size vectors must not exceed the bin capacity vector in each dimension, which is normalized to 1 without loss of generality. The VBP problem is also known as the multi-capacity BP problem in some literature [23], [24].

EUA Problem. In the EUA problem, each edge server corresponds to a bin with its available computing resources being the bin capacity. An app user corresponds to an item, whose computing resource requirement corresponds to the size of that item.

In this paper, we tackle the EUA problem from the app vendor's perspective. Since different users require various amounts of different computing resources [**?**], [25], [26], the amount of computing resources required by a user should be presented as a $d-$dimensional vector where each dimension represents a resource type (CPU, RAM, storage, etc.) instead of a single aggregate measure. This is also applied to the amount of computing resources available on edge servers. Therefore, the EUA problem can be modeled as a mixture of the VSBP problem and the VBP problem, hence being a variable-sized vector bin packing (VSVBP) problem. The EUA problem is $\mathcal{NP}$-hard since the classic BP problem, which is $\mathcal{NP}$-hard [22], is a special case of the VSVBP problem where all bins' sizes and all resources' requirement of users are identical.

4 Problem Formulation

Given a number of app users belong to a single app vendor using the same application, our objective is to maximize the total number of allocated app users and minimize the number of edge servers required to serve those users. We first introduce the relevant notations used in our model in Table 1. In the EUA problem, every user covered by any edge server must be allocated to an edge server unless all the servers accessible to the user have reached their maximum resource capacities. If a user cannot be allocated to any edge servers or is not positioned within the coverage of any edge servers, they will be directly connected to the app vendor's central cloud server.

TABLE 1: Notations

Notation	Description
$\mathcal{S} = \{s_1, s_2, ..., s_m\}$	a finite set of edge server s_i, where $i = 1, 2, ..., m$.
$\mathcal{D} = \{CPU, RAM, storage, bandwidth\}$	a set of computing resource types.
$C_i = \langle C_i^1, C_i^2, ..., C_i^d \rangle$	the capacity of an edge server s_i. C_i is a $d-$dimensional vector with each dimension $C_i^k, k = 1, ..., d$, representing the available amount of resource type $k \in \mathcal{D}$ on edge server s_i.
$\mathcal{U} = \{u_1, u_2, ..., u_n\}$	a finite set of user u_j, where $j = 1, 2, ..., n$.
$w_j = \langle w_j^1, w_j^2, ..., w_j^d \rangle$	$d-$dimensional vector representing the computing resource requirement of user u_j. Each vector component $w_j^k, k = 1, ..., d$ represents the amount of resource type k needed by user u_j on an edge server.
$\mathcal{U}_{s_i} \subset \mathcal{U}$	a set of users allocated to edge server s_i.
$\mathcal{S}_{u_j} \subset \mathcal{S}$	a set of user u_j's neighbor edge servers, i.e., edge servers that have user u_j in their coverage areas.
$cov(s_i)$	the coverage of edge server s_i.

In the classic BP problem, an item can be placed in any bin as long as the bin has sufficient remaining capacity. However, in our problem, each edge server covers a limited geographical region. Thus, an item (user) can only be placed in one of several specific bins (edge servers). A user u_j can be allocated to an edge server s_i only if it is located in server s_i's coverage area:

$$u_j \in cov(s_i), \forall u_j \in \mathcal{U}; \forall s_i \in \mathcal{S} \tag{1}$$

and the total resource requirements of all users allocated to an edge server must not exceed its available computing resources (Constraint (2)). Otherwise, the server will be overloaded, causing service disruptions or performance degradation. Take Fig. 1 for instance, the aggregated resource requirements of users u_1, u_2, and u_3 are $\langle 7, 11, 8, 10 \rangle$, which are greater than the available computing resources of edge server s_1 ($\langle 4, 6, 6, 10 \rangle$). Thus, allocating all those three users to this edge server violates the capacity constraint.

$$\sum_{u_j \in \mathcal{U}_{s_i}} w_j \preceq C_i, \forall s_i \in \mathcal{S} \tag{2}$$

Our primary objective is to maximize the number of users allocated to edge servers, which ensures the quality of service from the app vendor's perspective:

$$maximize \sum_{s_i \in S} |\mathcal{U}_{s_i}| \tag{3}$$

Our secondary objective is to find a users-to-edge-servers allocation such that the number of required edge servers is minimum:

$$minimize \left| \{s_i \in \mathcal{S} | \sum_{u_j \in \mathcal{U}_{s_i}} w_j > 0\} \right| \tag{4}$$

5 Approaches

5.1 Optimal Approach

In this paper, we address the EUA problem with two optimization objectives: *1) maximizing the number of allocated users* and *2) minimizing the number of required edge servers*, while satisfying the *proximity constraint* and *capacity constraint*. Accordingly, we solve the EUA problem with the Lexicographic Goal Programming (LGP) technique [27]. With the LGP technique, multiple optimization objectives are ranked by their levels of importance, or priorities. The problem solver will attempt to find an optimal solution that satisfies the primary objective and then proceed to find a solution for the next objective without deteriorating the previous objective(s). An LGP program can be solved as a series of connected linear programs. The LGP formulation of the EUA problem is as follows:

$$\text{maximize} \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} \tag{5}$$

$$\text{minimize} \sum_{i=1}^{m} y_i \tag{6}$$

$$\text{subject to: } x_{ij} = 0 \qquad \forall i, j \in \{i, j | u_j \notin cov(s_i)\} \tag{7}$$

$$\sum_{j=1}^{n} w_j^k x_{ij} \leq C_i^k y_i \quad \forall i \in \{1, ..., m\}; \forall k \in \{1, ..., d\} \tag{8}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad \forall j \in \{1, ..., n\} \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in \{1, ..., m\}; \forall j \in \{1, ..., n\} \tag{10}$$

$$y_i \in \{0, 1\} \qquad \forall i \in \{1, ..., m\} \tag{11}$$

x_{ij} and y_i are binary indicator variables such that

$$x_{ij} = \begin{cases} 1, & \text{if user } u_j \text{ is allocated to edge server } s_i. \\ 0, & \text{otherwise.} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{if edge server } s_i \text{ is used to serve users.} \\ 0, & \text{otherwise.} \end{cases}$$

Objective (5) maximizes the number of app users allocated to edge servers. Objective (6) minimizes the number of edge servers required to serve allocated users. Here, objective (5) is prioritized over objective (6). There are two groups of binary variables, i.e., x_{ij} (10) and y_i (11).

Proximity constraint (7): An edge server cannot serve users who are located outside its coverage area. An app user may be in the overlapping coverage area of multiple edge servers. For instance, user u_1 can be allocated to either edge server s_1 or s_3 in Fig. 1.

Capacity constraint (8): Each edge server s_i has an available resource capacity of $C_i = \langle C_i^1, C_i^2, ..., C_i^d \rangle$, a d-dimensional vector. The aggregated resource requirements of each resource type of all users allocated to an edge server must not exceed its available capacity. Take Fig. 1 for example, allocating users u_1 and u_2 to server s_1 is valid since $\langle 2, 4, 2, 4 \rangle \preceq \langle 4, 6, 6, 10 \rangle$.

Constraint family (9) ensures that every app user is allocated to at most one edge server. In other words, a user can be allocated to either an edge server or the app vendor's remote cloud server.

We can find an optimal solution to a small-scale instance of the EUA problem above with an Integer Programming solver, e.g., Gurobi[3] or IBM ILOG CPLEX[4]. In our experiments, we use IBM ILOG CPLEX Optimizer solver V12.8.0 (ilog.cplex[5] package in Java).

5.2 Most-Capacity-First Heuristic

Due to the $\mathcal{NP}$-hardness of the problem, finding an optimal solution will take a very long time in large-scale scenarios. This will be demonstrated in our experimental results presented in Section 6. Thus, to help app vendors allocate their users in large-scale scenarios, this section proposes MCF (MOST CAPACITY FIRST), an effective and efficient heuristic for finding a near-optimal solution to the EUA problem.

Given a set of edge servers $\mathcal{S}$ and a set of users $\mathcal{U}$ (lines 1-4), MCF efficiently and effectively allocates an app vendor's users to edge servers. Initially, all users are unallocated. In order to maximize the number of users allocated, MCF first sorts the set of users $\mathcal{U}$ in ascending order of users' computing resource requirements (line 5). The computing resource requirement of a user is a multi-dimensional vector with each component being a resource type. Since different resource types (CPU, RAM, storage, or bandwidth) have different scales, we can sort the computing resource requirements by normalizing all resource types' requirements by maximum norm (assuming all resource types are equally important), then calculate the Euclidean norm over those resource types. App vendors can also apply other methods [28] as they see fit to sort users or servers by the amount of computing resources.

Heuristic 1 Most Capacity First (MCF)

1: **initialization:**
2: a set of edge servers $\mathcal{S}$ and a set of users $\mathcal{U}$
3: all users $u_j, \forall u_j \in \mathcal{U}$, are unallocated
4: **end initialization**
5: sort $\mathcal{U}$ in ascending order of computing resource requirements (i.e., low-demanding users are prioritized, being the first to be allocated)
6: **for** each user $u_j \in \mathcal{U}$ **do**
7: $\quad \mathcal{S}_{u_j} \triangleq$ user u_j's neighbor edge servers;
8: $\quad \mathcal{S}_{u_j}^{active} \triangleq$ user u_j's active neighbor edge servers;
9: $\quad$ **if** $\mathcal{S}_{u_j}^{active} \neq \emptyset$ **then**
10: $\quad\quad$ allocate user u_j to an edge server $s_i \in \mathcal{S}_{u_j}^{active}$ which has the most available capacity
11: $\quad$ **else**
12: $\quad\quad$ allocate user u_j to an edge server $s_i \in \mathcal{S}_{u_j}$ which has the most available capacity
13: $\quad$ **end if**
14: $\quad$ **if** s_i cannot be decided **then**
15: $\quad\quad$ allocate user u_j to the central cloud server
16: $\quad$ **end if**
17: **end for**

Next, MCF allocates users one by one (lines 6-14) in the order of their appearance in the list $\mathcal{U}$ sorted above. Since the list of users has been sorted in ascending order of resource requirements, users who have lower requirements will be allocated before users who have higher requirements. This helps maximize the number of users allocated because allocating high-demanding users first would exhaust edge servers' resources rather quickly. For each user, MCF retrieves the set of its neighbor edge servers (servers that have the user in their proximity - line 7) and the set of its active neighbor edge servers (servers that have already been used to serve users, or are serving users - line 8). To minimize the number of required edge servers, MCF attempts to allocate the user to one of the active servers first (line 9), instead of using a new edge server. Out of all active edge servers, one with the most available capacity will be chosen to serve the user (line 10). In this way, it will be most likely to have sufficient capacity to accommodate other users later on. If there is no current active server (line 11), MCF will allocate the user to the neighbor edge server which has the most available capacity (line 12). Note that allocating a user to an edge server must not violate the capacity constraint in any case. If all neighbor edge servers of the user have reached their maximum capacity, the user will be directly connected to the app vendor's central cloud server (lines 14-16), which is not desirable. MCF completes once all users have been attended to.

Time Complexity Analysis. As discussed at the end of Section 3, the EUA problem is $\mathcal{NP}$-hard. It is intractable to find optimal solutions to large-scale EUA scenarios that involve a large number of app users and/or edge servers. MCF is a practical option in such scenarios for its high efficiency. Here, we analyze the worst-case time complexity

3. www.gurobi.com/
4. www.ibm.com/analytics/cplex-optimizer/
5. www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.cplex.help/refjavacplex/html/ilog/cplex/package-summary.html

of the proposed heuristic (MCF - Heuristic 1). The running time of MCF consists of: 1) iterating through all n users (Line 6), which costs $\mathcal{O}(n)$, and 2) sorting a maximum of m edge servers to identify the edge server that has the most available capacity (Line 10 or 12), which costs $\mathcal{O}(m\log(m))$. Therefore, the time complexity of this block (Lines 7-13) is $\mathcal{O}(nm\log(m))$, which is generally more complex than Line 5. Hence, the overall time complexity of MCF is $\mathcal{O}(nm\log(m))$. Note that m is the worst-case scenario; in practice, each user is covered by a much smaller number of edge servers or base stations (i.e., 32 base stations in theory as specified by ETSI [29], or 15 base stations according to the dataset used in our experiments in Section 6). As experimentally demonstrated in Section 6.3.2, the running time of MCF is around 1-2 ms in all scenarios.

Given the low running time of MCF, app vendors are able to continuously run it to react to user movement since it allocates app users one-by-one to edge servers. Specifically, when a user moves out of the coverage area of the edge server serving it, it will be disconnected from the edge server; the occupied computing resources on that edge server will be released; then MCF will consider it as a new user and allocate it to a new edge server following the rules defined in Heuristic 1, Lines 6-14. This is possible as long as switching users from one edge server to another does not incur extra costs, or if the extra costs are trivial. Such extra costs may include the migration cost (e.g., the cost of moving user data across edge servers) or reconfiguration cost (e.g., some services might require reconfiguration to serve new users) [30]. In some use cases, those extra costs could be fairly trivial. Take video streaming for example [31], where videos encoded with different resolutions are cached on edge servers so that a user can access them with low latency. Switching the user across edge servers would only require a small amount of data to be transferred, e.g., which video the user is watching, the resolution of the video, and the position in the video where the user left off. For applications and services where the extra costs are considerable, the new costs will need to be modeled and integrated into the objective functions. The heuristic will thus need to be modified accordingly.

6 Experimental evaluation

We have performed a series of experiments on a real-world dataset to evaluate the performance of our approaches against other baseline and state-of-the-art approaches.

6.1 Performance Benchmark

Our optimal approach (Section 5.1), referred to as Optimal hereafter, and MCF heuristic (Section 5.2) are compared to four representative approaches, namely a Greedy baseline, a Random baseline, and two state-of-the-art approaches for solving the EUA problem:

- *Greedy*: This approach allocates each user to the edge server with the most available capacity, regardless of the server's active status. Users are allocated in no particular order.
- *Random*: This approach allocates each user to a random edge server available. Users are allocated in no particular order.
- *ICSOC19*: [19] proposes two approaches to solve the user allocation problem so that the total users' QoE is maximized. The QoS level, or the computing resource requirement, of each user can be flexibly adjusted. We solve a slightly different problem where each user has a fixed QoS level. Their proposed optimal approach will be used in our experimental evaluation.
- *TPDS20*: [**?**] proposes a game-theoretic approach to solve the user allocation problem with the objective of maximizing the number of allocated users and minimizing system costs, measured by the amount of computing resources needed to serve users and the penalty of having unallocated users.

ICSOC19 and TPDS20 are chosen as the benchmark state-of-the-art approaches as they solve the same problem as ours – user allocation, and their objectives indirectly imply the maximization of the number of allocated users. TPDS20 also implies the minimization of the number of required edge servers. All experiments are written in Java and conducted on a Windows machine equipped with Intel Core i5-7400T processor (4 CPUs, 2.4GHz) and 8GB RAM.

6.2 Experimental Settings

The experiments are conducted on the EUA dataset[6] [1], which contains the geographical locations of end-users and all cellular base stations in Australia. This dataset was also used in [19] and [**?**] to evaluate ICSOC19 and TPDS20.

Edge servers: To capture the characteristics of a 5G environment [32], we simulate a highly dense urban area of 1.8 km^2 covered by 125 base stations, each equipped with an edge server. The coverage radius of each edge server is randomly generated within 100-150m. The initial capacities of edge servers are randomly generated by following a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where μ is the average capacity of each resource type in $\mathcal{D}$, and the standard deviation $\sigma = 10$ for all conducted experiments. Since a normal distribution might contain negative numbers, any negative amount of computing resources generated is rounded up to 1.

Edge users: We assume that there are three possible types of resource requirements, $w_j \in \{\langle 1,2,1,2\rangle, \langle 2,3,3,4\rangle, \langle 5,7,6,6\rangle\}$, $\forall u_j \in \mathcal{U}$, and $\mathcal{D} = \{CPU, RAM, storage, bandwidth\}$. The computing resource requirement of each user is uniformly randomly selected. We select those three resource requirement levels as representative in our experiments since we have conducted experiments with other types of resource requirement settings, where the difference between resource requirements is large (e.g., $\{\langle 1,1,1,1\rangle, \langle 4,5,6,6\rangle, \langle 7,9,10,7\rangle\}$), or where the resource requirements are identical, or skewed (e.g., $\{\langle 3,2,3,2\rangle, \langle 2,3,2,3\rangle, \langle 1,2,1,3\rangle\}$), which all returned similar results (almost identical, or just marginally different).

To comprehensively analyze the performance of our approaches in various EUA scenarios, we conduct a series of experiments with different varying parameters, including the number of users, the number of edge servers, and edge

6. www.github.com/swinedge/eua-dataset

servers' capacities (μ as defined above). Table 2 summarizes the settings of the experiments, which will be discussed in the next section. Each experiment is repeated 100 times to obtain 100 different user distributions, and the results are then averaged. This allows extreme cases, such as overly dense or sparse user/server distributions, to be neutralized. To evaluate the performance of the approaches in achieving the optimization objective, we compare the number of allocated users and required edge servers achieved by the six approaches, and also the average number of users allocated per required edge server. The efficiency will also be evaluated by the CPU time taken to solve the problem.

TABLE 2: Experimental Settings

	Users	Edge servers	Available resources (μ)
Set #1	$100, ..., 1000$	50%	35
Set #2	500	$10\%, ..., 100\%$	35
Set #3	500	50%	$30, 35, ..., 75$

6.3 Experimental Results

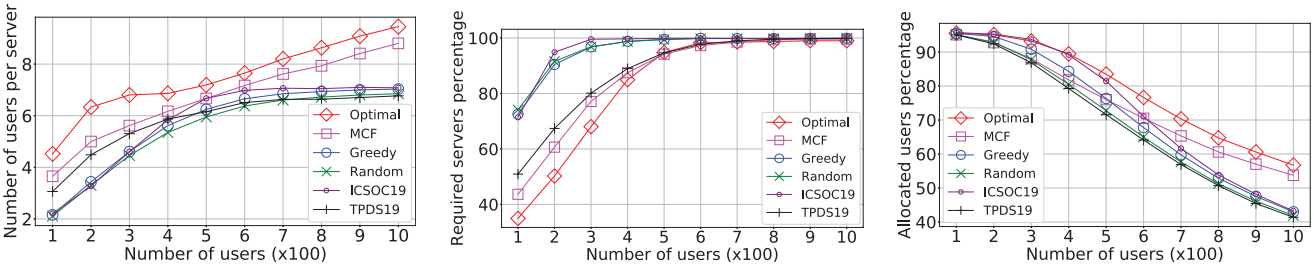
Figs. 2, 3, and 4 demonstrate the effectiveness of all the approaches in experiment Sets #1, #2, and #3, respectively, in terms of 1) the percentage of users allocated, the higher the better (in sub-figures (c)), 2) the percentage of edge servers needed to serve those users, the lower the better (in sub-figures (b)), and 3) we additionally measure the average number of users allocated per required edge server, the higher the better (in sub-figures (a)). In general, Optimal, being the optimal approach, clearly achieves the best performance compared to all other approaches across all experiments – being able to allocate the most number of users to the fewest number of edge servers. This comes at the cost of its very high computational overhead and it is thus inapplicable in large-scale scenarios, where low latency is critical. MCF outperforms all other baseline and state-of-the-art approaches. The efficiency of MCF is demonstrated in Figs. 5 and 6, measured by its computation time.

6.3.1 Effectiveness

Experiment Set #1. In this set of experiments, the number of users varies from 100 to 1,000 in steps of 100. The number of edge servers is fixed at 50% of all edge servers in the simulated area. In Fig. 2a, as the number of users increases, the average number of users allocated per required edge server achieved by Optimal also increases, closely followed by MCF. While the average number of users per required edge server achieved by Optimal and MCF increases linearly, the performance of the other four approaches starts to plateau at some point, e.g., 600 users in our experiments. This depicts the ineffectiveness of the other approaches in large-scale scenarios with a great number of users. With regard to the number of edge servers required (Fig. 2b), the approaches are divided into two groups based on their performance. The first group, which includes ICSOC19, Greedy, and Random, requires far more edge servers than the second group, which consists of Optimal, MCF, and TPDS20. Optimal requires the fewest number of edge servers, followed by MCF, then TPDS20. From 600 users onwards, every approach requires 100% number of edge servers since there is a large number of users now. Thus, all available computing resources need to be utilized. Fig. 2c shows the percentage of users allocated. We can observe a decreasing trend here. Since the amount of computing resources is fixed, introducing more users will increase the number of users who cannot be allocated to any edge server. From 100 to 500 users, MCF allocates slightly fewer users than Greedy and ICSOC19. However, in those cases, Greedy and ICSOC19 require far more edge servers to serve those users, which results in a lower average number of users per required server in general as shown in Fig. 2a. Other than those cases, MCF allocates considerably more users than Greedy, Random, ICSOC19, and TPDS20. ICSOC19 performs poorly (almost as bad as Greedy and Random) because it aims to maximize all users' overall QoE, which in turn needs to consume as much computing resources as possible, hence a high number of required edge servers. In the meantime, ICSOC19 is also very computationally expensive since it aims to find the optimal solution to an $\mathcal{NP}$-hard problem. Its low effectiveness and low efficiency can be observed in other experiment sets as well. While being suitable for maximizing users' overall QoE, we can see that ICSOC19 is not effective in minimizing the number or required edge servers.

Experiment Set #2. In this experiment set, we vary the number of edge servers available to serve users, from 10% to 100% in steps of 10% (Fig. 3). As more edge servers become available, more computing resources are available to serve users, which eventually increases the number of allocated users (Fig. 3c). In this aspect, the difference between all the approaches is marginal. However, when it comes to the number of edge servers needed (Fig. 3b), Optimal, MCF, and TPDS20 significantly outperform other approaches, requiring much fewer edge servers to serve users. ICSOC19, Greedy, and Random require almost all edge servers in all settings. Meanwhile, the percentage of edge servers required by Optimal, MCF, and TPDS20 rapidly decreases as the number of available edge servers increases. This demonstrates the effectiveness of those approaches in utilizing the given resources. Overall, the average number of allocated users per required server (Fig. 3a) is higher for Optimal and MCF (closely followed by TPDS20 from 40% onwards). Greedy, Random, and ICSOC19 fail to utilize the increasing number of edge servers, hence the downward trend. In some cases (20% - 40% for Optimal, 20% - 60% for MCF), the average number of allocated users per required edge server decreases because the cost of using edge servers (the number of required edge servers) outweighs the benefit of serving more users. Regardless, Optimal and MCF still beat other approaches.

Experiment Set #3. In this experiment set, we vary the amount of average available computing resources on each server. Similar to experiment Set #2, increasing edge servers' capacities eventually increases the total number of allocated users (Fig. 4c) and the average number of allocated users per required server (Fig. 4a). It also generates more room for resource utilization. In Fig. 4b, Optimal, MCF, and TPDS20 demonstrate the ability to utilize the given computing resources by the decreasing percentage of required edge servers. MCF, again, requires fewer edge servers than TPDS20. When μ is increasing, the capacity of each edge

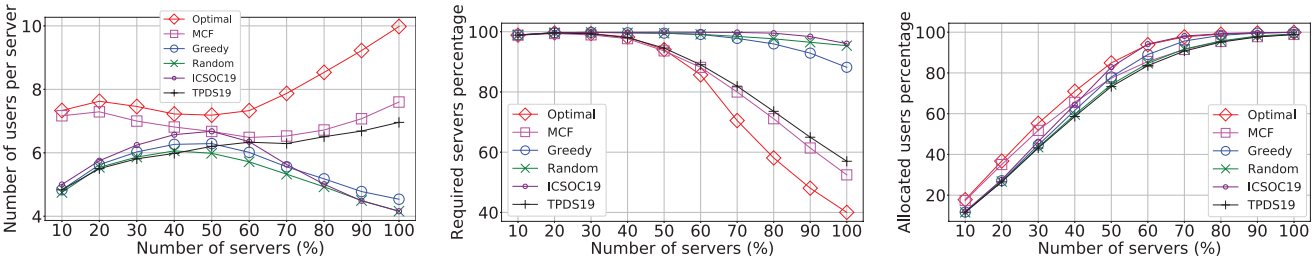


(a) Number of users vs. Average number of users per required edge server

(b) Number of users vs. Percentage of edge servers required

(c) Number of users vs. Percentage of users allocated

Fig. 2: Experimental results of experiment Set #1 (varying number of users)

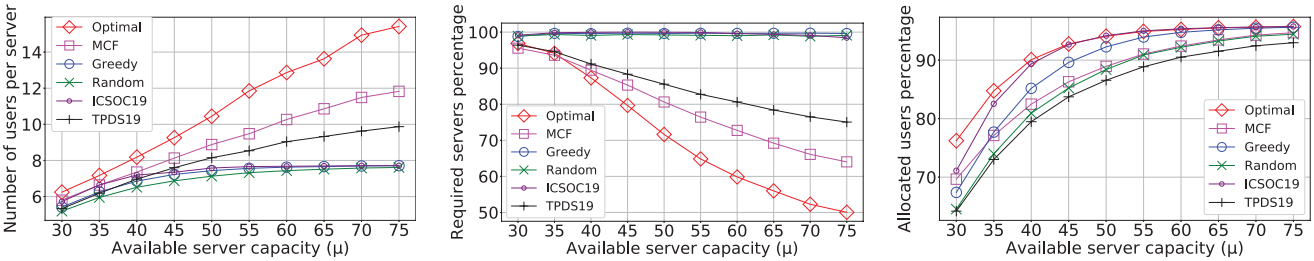


(a) Number of edge servers vs. Average number of users per required sdge server

(b) Number of edge servers vs. Percentage of edge servers required

(c) Number of edge servers vs. Percentage of users allocated

Fig. 3: Experimental results of experiment Set #2 (varying number of edge servers)



(a) Edge server capacity vs. Average number of users per required edge server

(b) Edge server capacity vs. Percentage of edge servers required

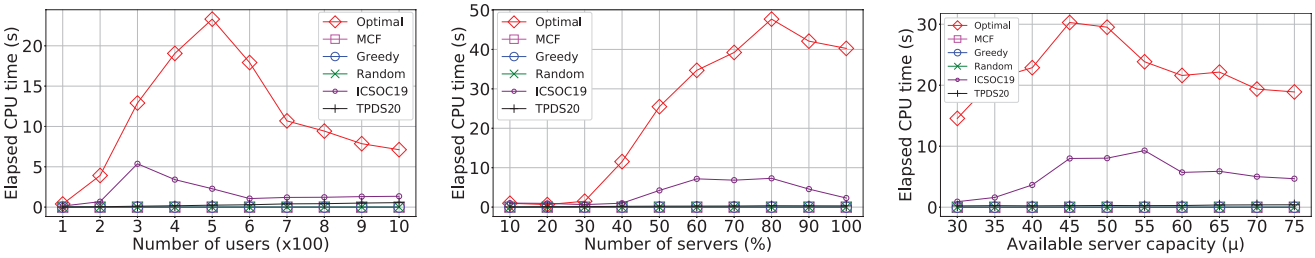
(c) Edge server capacity vs. Percentage of users allocated

Fig. 4: Experimental results of experiment Set #3 (varying edge server' capacity)

server becomes increasingly redundant, rendering an increasing number of "unnecessary" edge servers, hence the decrease in the percentage of required edge servers.

6.3.2 *Efficiency*

Fig. 5 depicts the efficiency of all the approaches, measured by the average CPU execution time taken to solve an instance of the EUA problem. Optimal is the most inefficient approach that might take up to 50 seconds to find an optimal user allocation solution. The elapsed CPU time of Optimal increases considerably as we increase the size of the EUA problem by adding more users (Set #1, Fig. 5a), more edge servers (Set #2, Fig. 5b), and more edge server capacity (Set #3, Fig. 5c). When it reaches a threshold, the elapsed CPU time of Optimal decreases at a slower rate than it increases. This threshold is determined by the number of users to be allocated and the amount of computing resources (available edge server capacity or available edge servers). To be specific, in Fig. 5a, the CPU time starts to decrease at 500 users. This happens because when the number of users exceeds 500, a newly generated user tends to be positioned at the exact location of an existing user. Thus, the IBM CPLEX solver can base its decisions to be made for the new user on the decisions made for the existing user. As a result, we can see the elapsed CPU time is almost symmetrical around that threshold (500 users). In Figs. 5b and 5c, the elapsed CPU time decreases from 80% of the total number of edge servers and from $\mu = 45$ onward,

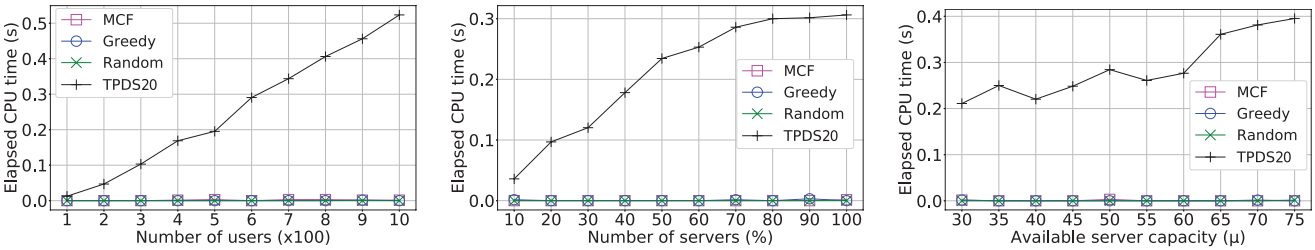


(a) Elapsed CPU time vs. Number of users (Set #1)

(b) Elapsed CPU time vs. Number of edge servers (Set #2)

(c) Elapsed CPU time vs. Edge server capacity (Set #3)

Fig. 5: CPU time consumption in experiment Sets #1, #2, and #3



(a) Elapsed CPU time vs. Number of users (Set #1)

(b) Elapsed CPU time vs. Number of edge servers (Set #2)

(c) Elapsed CPU time vs. Edge server capacity (Set #3)

Fig. 6: CPU time consumption in experiment Sets #1, #2, and #3 without Optimal and ICSOC19

respectively. This occurs because after those thresholds, the whole MEC system is likely to have sufficient computing resources to serve a greater number of users without having to consider many other possible allocation solutions, hence less time required to decide an optimal solution.

Due to the extreme inefficiency of Optimal and ICSOC19, Fig. 5 does not fully demonstrate the efficiency of other approaches. Therefore, Optimal and ICSOC19 have been excluded in Fig. 6 so that the efficiency of other approaches can be visualized better. In all experiment sets (Figs. 6a, 6b, and 6c), TPDS20 is two to three orders of magnitude slower than other approaches, and its execution time increases roughly linearly with the increases in any experimental parameters. The rationale of this lies in the algorithm design of TPDS20, which is an iterative mechanism that might involve hundreds of iterations depending on the scale of the problem. MCF, on the other hand, only takes around 1-2 ms to solve the problem in any experimental setting. This demonstrates the scalability of MCF.

Generally speaking, MCF has been demonstrated to be highly effective and efficient, outperforming all the baseline and state-of-the-art approaches. TPDS20, while being suitable for solving that particular problem studied in [**?**], is not suitable when it comes to solving the EUA problem introduced in this paper. After all, game-theoretic approaches have not been seen in the current literature as a method for solving the bin packing problem.

6.3.3 *Statistical Analysis*

In this paper, each experiment is repeated 100 times to obtain 100 different random user distributions. To determine whether the approaches in comparison (Optimal, MCF, Greedy, Random, ICSOC19, and TPDS20) are really different when applied to all possible user distributions (of the relevant size), we perform one-sided Wilcoxon signed-rank tests [33] with a significance level of 0.01 on all three experiment sets. The detailed statistical results can be found in Appendix A of the supplemental file. In summary, the Wilcoxon signed-rank tests indicate that the difference between the effectiveness of MCF and other baseline approaches under the majority of experimental settings are statistically significant at the $\alpha = 0.01$ level. The statistical results are in line with the experimental results discussed above. With regard to the efficiency, the execution time of MCF is consistent at around 1-2 ms so we do not perform a statistical test for that.

6.4 Threats to Validity

Threats to Construct Validity. To mitigate the threats to construct validity, we evaluate the performance of our approach against several baseline approaches (Greedy and Random), well-known approaches for solving the bin packing problem (demonstrated in Section 7.2), and state-of-the-art approaches (TPDS20 and ICSOC19). Furthermore, we conducted experiments with three varying parameters as described in Table 2 to simulate different edge computing scenarios in the real world. In this way, we could reliably evaluate our approach through both comparisons with the other approaches and also the impacts of each varying experimental parameter on our approach.

Threats to External Validity. A major threat to external validity is whether our findings can be generalized to other application domains in edge computing. To minimize this

threat, we first use a generic dataset. Since different application domains might have different factors that could impact the experimental results, such as the density and distribution of edge servers and users, our approach is evaluated across various sizes and complexity, which is controlled by varying the number of users, the number of edge servers as well as their available computing capacities. This helps increase the generalizability of our results. Different users' computing requirements have also been tested, which produced similar results presented in this paper.

Threats to Internal Validity. A threat to the internal validity of our work is the comprehensiveness of our experiments and whether or not the results are not biased by the experimental parameter settings. To mitigate this threat, we carry out extensive experiments with systematically selected parameters. The three experimental parameters (Table 2) are the three representative ones that directly impact the outcomes of the approaches. Additionally, for each experiment set, we experiment with 100 different user distributions, randomly selected from the pool of users to eliminate the potential bias caused by highly special scenarios, such as overly dense or sparse distributions.

Threats to Conclusion Validity. To mitigate the threats to conclusion validity, we conduct a comprehensive evaluation of the proposed heuristic against other approaches that cover many different scenarios, varying in both size and complexity. The results are averaged over 100 runs of the same experimental setting. After that, a statistical test is performed to validate the difference between the proposed heuristic and other baseline approaches.

7 Related work

Edge computing is a natural extension of cloud computing with regard to the network topology and infrastructure deployment, where the architecture is more geographically distributed compared to cloud computing. This new architecture pushes cloud computing resources closer to end-users. Barcelona in Spain is one of the first cities implementing edge computing with many applications, including power monitoring in public spaces, event-based video streaming, traffic analysis, and connectivity on-demand [**?**]. There are more than 3,000 edge servers deployed across the city serving thousands of IoT devices. The sheer number of edge servers and end-devices, with the horizontal scaling nature of edge computing, leads to the need for effective and efficient solutions to many different research problems faced by app vendors, including the user allocation problem.

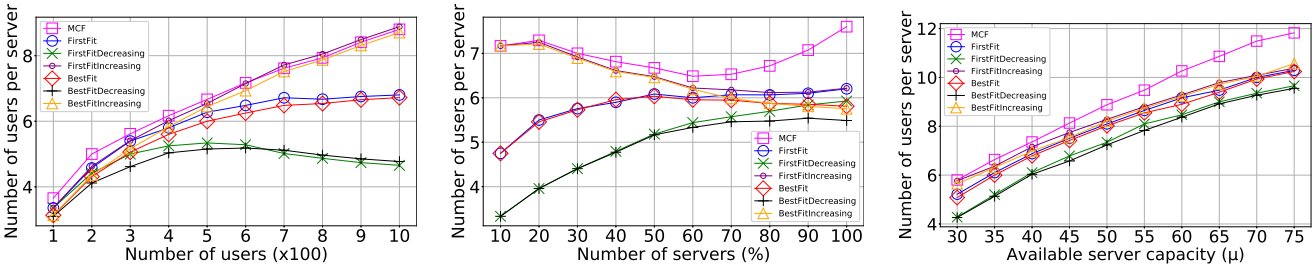
7.1 User Allocation

In an edge computing environment, an app vendor can deploy its applications and services on edge servers so that users can access those applications and services with minimum latency. From an app vendor's perspective, the user allocation problem is the problem of how to allocate its users to proper edge servers so that some optimization objectives are satisfied. We first introduced the user allocation problem in our previous work [1]. However, the extreme inefficiency of the proposed approach has led us to this current work. [34] studies the cloudlet placement and users-to-cloudlets allocation in wireless metropolitan networks. In their scenario, users are connected to access points, which might or might not have a cloudlet. They group all users who are connected to the same access point and then turn it into the problem of allocating access points to cloudlets from an edge infrastructure provider's perspective. We, on the other hand, tackle the user allocation problem from an app vendor's perspective in the edge computing environment and allocate users to edge servers individually. [35] studies the service placement problem that takes into account user mobility. The users-to-edge-servers allocation is assumed to be automatically handled by the edge infrastructure provider. [13] also tackles the resource allocation for edge computing, which allocates edge server computing resources to multiple competing services owned by different app vendors at the network edge. We take the next step and address the user allocation problem, which allocates the edge server computing resources owned by a single app vendor to its users.

In [36], [37], the authors assume that each small geographical area will only receive coverage from a single edge server, or each server covers a region exclusively with other servers. This is unlikely to happen in any practical mobile edge computing situations, where different edge servers' coverages might partially overlap to avoid non-service areas [11], [19]. [18] considers the user mobility scenario where users can move from one place to another, which requires reallocating users among edge servers. Their user allocation approach aims at maximizing the number of users while minimizing the number of reallocations. This does not help minimize the number of required servers, which is one of our optimization objectives. [**?**] proposes a game-theoretic user allocation approach that minimizes system costs, measured by the amount of computing resources needed to serve users and the penalty of having unallocated users. This approach also indirectly minimizes the number of required edge servers so we have compared it with our approach in Section 6.

Computation offloading [11], [17], [38] is an important research track in edge computing that shares some similarities with the user allocation problem. Nevertheless, those two problems are differentiated by several essential characteristics. In the computation offloading problem, a user generates a series of computation tasks, which can be partly executed on its local device and edge servers (partial offloading), or completely on edge servers or remote clouds (full offloading) [39]. A computation task usually has a single-dimensional resource requirement (CPU cycles) [11], [17], [40]. On the other hand, in the user allocation problem, an app vendor needs to dedicate multiple types of resources to serve a user on an edge server [**?**], [25], [26]. In some works [11], [17], users are assumed to be pre-allocated to edge servers before proceeding to the task offloading phase. Furthermore, a user in the user allocation problem must be allocated to a server, either an edge server or a cloud server, instead of being able to partially offload its computation tasks, or share computation tasks among edge servers.

Cloud task scheduling is also a research area that shares some similarities with the EUA problem. Nevertheless, from the app user's perspective, the virtual machines or clouds in the same data center share the same reachability while an



(a) Average number of users per required edge server vs. Number of users (Set #1)

(b) Average number of users per required edge server vs. Number of edge servers (Set #2)

(c) Average number of users per required edge server vs. Edge server capacity (Set #3)

Fig. 7: Comparison of several classic approximation algorithms for the bin packing problem

app user can only access an edge server that covers it. This proximity constraint is one of the unique characteristics in the edge computing environment, which has been widely acknowledged in many other studies of edge computing [**?**], [11], [18], [19].

7.2 Bin Packing

Bin Packing is a classic combinatorial $\mathcal{NP}$-hard optimization problem widely applied in different disciplines such as operation research, or computer science (multiprocessor scheduling [41], cloud task and resource allocation [42], [43]). There are many variations of this problem, e.g., packing by weight or by cost [**?**], multi-dimensional BP [44], dynamic BP [45], and a lot more variants with different modified conditions, constraints, and assumptions to model different problems. The latest attempt at exactly solving the BP problem involves constraint programming. Shaw [46] proposes a new dedicated constraint based on a set of pruning and propagation rules, which is later on implemented in IBM CPLEX [47], which is the tool used in this paper to search for optimal solutions to EUA problems.

Along with exact algorithms, various approximation algorithms have also been introduced to solve different variants of the classic BP problem. Most of the proposed approximation algorithms are designed for different special cases or variants of the classic BP problem, such as splittable, small items (relative to the size of a bin) [48], variable-sized items with identical bins [28], minimizing the total used bins load with only two bin sizes [49]. Surveys on approximation algorithms for BP problems can be found in [47], [50]. BP is a straightforward problem so any improvement in online solutions would require a special assumption as mentioned above. The authors of [51] state that the theoretical analysis of variable-sized BP remains open even in the case of only three different bin sizes. In an MEC environment, edge servers are most likely to have more than three different sizes (capacities). Therefore, to properly evaluate our approaches, we also compare our approaches with some representative approximation algorithms for solving the BP problem [51], including First Fit, First Fit Decreasing, First Fit Increasing, Best Fit, Best Fit Decreasing, Best Fit Increasing (Fig. 7). Next Fit and Next Fit Decreasing are not suitable for the EUA problem since it allows only one open bin at all times. Our proposed MCF is essentially a variant of Worst Fit Increasing, which prioritizes already-active edge servers and is adapted to multi-dimensional computing resource requirements.

As demonstrated by Figs. 7a and 7c, the increasing variants, which allocate users with low resource requirements first, outperform other approaches, especially the decreasing variants. This highlights the importance of the order of users being allocated. In all experimental settings, especially Set #2 (Fig. 7b), MCF outperforms all other approaches in comparison. More experimental results and statistical tests can be found in Appendix B of the supplemental file. [52] models the machine reassignment problem also as a VSVBP problem. Its proposed heuristic is a generalization of First Fit Decreasing and Best Fit Decreasing, which have been shown above to be not suitable for the EUA problem.

8 Conclusion and Future Work

Further complementing the conventional cloud computing, edge computing is a promising distributed computing architecture that is expected to deliver many new genres of services and applications, especially those that require low-latency connection and real-time decision making. This comes with many new challenges of which user allocation is one of them. When an edge computing environment scales up, this problem becomes intractable to solve in an efficient manner due to its $\mathcal{NP}$-hardness. Therefore, we propose MCF (Most Capacity First) – a simple yet highly effective and efficient heuristic, to solve the user allocation in large-scale scenarios, aiming to allocate as many users to as few edge servers as possible. Our experiments on a real-world dataset demonstrate the performance of our approach against the baseline and state-of-the-art approaches.

Being a new problem and has not been studied extensively, there are many possible directions for future work. Edge servers' performance degradation caused by the interference of workloads, or differentiated user workload patterns, may occur in some applications. This could potentially impact the performance and the available resources of an edge computing system, hence must be considered in future studies. Furthermore, it is possible that some users might not use the entire resources allocated to them during their runtime, leading to an under-utilized edge computing infrastructure. We can thus investigate the scenario where the runtime resource consumption might noticeably differ

from the resources allocated during the allocation process. One can also consider a dynamic scenario where users come and go over time or users can move across edge servers, which may incur extra costs such as service reconfiguration cost or migration cost. Those costs, if available, must be incorporated into the approach proposed in this paper.

ACKNOWLEDGMENTS

This research is partly funded by Australian Research Council Discovery Project grants DP170101932, DP180100212, and Laureate Fellowship FL190100035.

REFERENCES

[1] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proceedings of International Conference on Service-Oriented Computing*. Springer, 2018, pp. 230–245.

[2] P. Cerwall, A. Lundvall, P. Jonsson, S. Carson, R. Möller, P. Jonsson, S. Carson, P. Lindberg, K. Öhman, I. Sorlie, R. Queirós, F. Muller, L. Englund, M. Arvedson, and A. Carlsson, "Ericsson mobility report," *Ericsson, Stockholm*, 2018. [Online]. Available: www.ericsson.com/en/mobility-report/reports/november-2018

[3] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proceedings of IEEE Symposium on Computers and Communications*. IEEE, 2012, pp. 59–66.

[4] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[5] B. Varghese, N. Wang, D. S. Nikolopoulos, and R. Buyya, "Feasibility of fog computing," *arXiv preprint arXiv:1701.05451*, 2017.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[8] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[10] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[11] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proceedings of IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[12] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[13] D. T. Nguyen, L. B. Le, and V. Bhargava, "Price-based resource allocation for edge computing: A market equilibrium approach," *IEEE Transactions on Cloud Computing*, pp. 515–529, 2018.

[14] T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1027–1034, 2011.

[15] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, 2015.

[16] "Resource management in clouds: Survey and research challenges, author=Jennings, Brendan and Stadler, Rolf," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[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] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, H. Liu, Y. Qin, and P. Chen, "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *Proceedings of IEEE International Conference on Web Services*. IEEE, 2019, pp. 91–98.

[19] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *Proceedings of International Conference on Service-Oriented Computing*. Springer, 2019, pp. 86–101.

[20] Y. Lin and H. Shen, "CloudFog: Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service," *IEEE Transactions on Parallel and Distributed Systems*, no. 2, pp. 431–445, 2017.

[21] J. Smith, "The Mobile Gaming Report: Market size, the fee-to-play model, and new opportunities to market and monetize," Business Insider Intelligence, Tech. Rep., 2016. [Online]. Available: www.businessinsider.com/the-mobile-gaming-report-market-size-the-free-to-play-model-and-new-opportunities-to-market-and-monetize

[22] M. R. Garey and D. S. Johnson, *Computers and intractability*. WH Freeman New York, 2002, vol. 29.

[23] W. Leinberger, G. Karypis, and V. Kumar, "Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints," in *Proceedings of International Conference on Parallel Processing*. IEEE, 1999, pp. 404–412.

[24] H. Hallawi, J. Mehnen, and H. He, "Multi-capacity combinatorial ordering GA in application to cloud resources allocation and efficient virtual machines consolidation," *Future Generation Computer Systems*, vol. 69, pp. 1–10, 2017.

[25] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, 2019.

[26] M. Aazam, M. St-Hilaire, C.-H. Lung, and I. Lambadaris, "MeFoRE: QoE based resource estimation at fog to enhance QoS in IoT," in *Proceedings of International Conference on Telecommunications*. IEEE, 2016, pp. 1–5.

[27] C. Romero, *Handbook of critical issues in goal programming*. Elsevier, 2014.

[28] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *Microsoft Research*, 2011.

[29] ETSI, "Universal Mobile Telecommunications System (UMTS); Requirements for Support of Radio Resource Management (FDD) (3G TS 25.133 version 3.1.0 Release 1999)," Technical Report, Tech. Rep., 2000. [Online]. Available: www.etsi.org/deliver/etsi_ts/125100_125199/125133/03.01.00_60/ts_125133v030100p.pdf

[30] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "MOERA: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1843–1856, 2018.

[31] C. Liang, Y. He, F. R. Yu, and N. Zhao, "Enhancing video rate adaptation with mobile edge computing and caching in software-defined mobile networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 10, pp. 7013–7026, 2018.

[32] W. H. Chin, Z. Fan, and R. Haines, "Emerging technologies and research challenges for 5G wireless networks," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 106–112, 2014.

[33] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 2003.

[34] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[35] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[36] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 16, p. e3975, 2017.

[37] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Proceedings of International Conference on Distributed Computing Systems*. IEEE, 2017, pp. 1281–1290.
[38] N. Fernando, S. W. Loke, and W. Rahayu, "Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 329–343, 2016.
[39] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
[40] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.
[41] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, no. 1, pp. 1–17, 1978.
[42] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proceedings of ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 2014, pp. 2–11.
[43] C. Li and X. Tang, "On fault-tolerant bin packing for online resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, 2019.
[44] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 837–851, 2004.
[45] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Dynamic bin packing," *SIAM Journal on Computing*, vol. 12, no. 2, pp. 227–258, 1983.
[46] P. Shaw, "A constraint for bin packing," in *Proceedings of International Conference on Principles and Practice of Constraint Programming*. Springer, 2004, pp. 648–662.
[47] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.
[48] Y. Azar, I. R. Cohen, A. Fiat, and A. Roytman, "Packing small vectors," in *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2016, pp. 1511–1525.
[49] S. S. Seiden, R. Van Stee, and L. Epstein, "New bounds for variable-sized online bin packing," *SIAM Journal on Computing*, vol. 32, no. 2, pp. 455–469, 2003.
[50] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, vol. 24, pp. 63–79, 2017.
[51] E. G. Coffman Jr, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Survey and classification," in *Handbook of Combinatorial Optimization*. Springer, 2013, pp. 455–531.
[52] M. Gabay and S. Zaourar, "Variable size vector bin packing heuristics - Application to the machine reassignment problem," Sep. 2013, working paper or preprint. [Online]. Available: www.hal.archives-ouvertes.fr/hal-00868016

Phu Lai received his MSc degree in Information Technology in 2017 and is currently working toward a PhD degree at Swinburne University of Technology, Australia. His research interests include software engineering, cloud computing and edge computing.

Qiang He received the first PhD degree from Swinburne University of Technology (SUT), Australia, in 2009 and the second PhD degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2010. He is a lecturer at Swinburne University of Technology. His research interests include software engineering, cloud computing, services computing, big data analytics, and green computing. More details about his research can be found at www.sites.google.com/site/heqiang/.

John Grundy is the Senior Deputy Dean for the Faculty of Information Technology and a Professor of Software Engineering at Monash University. He holds the BSc(Hons), MSc and PhD degrees, all in Computer Science, from the University of Auckland. He is a Fellow of Automated Software Engineering, Fellow of Engineers Australia, Certified Professional Engineer, Engineering Executive, Member of the ACM and Senior Member of the IEEE. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at www.sites.google.com/site/johncgrundy/.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia, in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Mohamed has more than 15 years of the software industry, research, and teaching experience. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of the software development department at Microtech. More details about his research can be found at www.sites.google.com/site/mohamedalmorsy/.

John Hosking is Dean of Science at the University of Auckland and Adjunct Professor of Computer Science at the ANU. His research interests are primarily in the Software Engineering/Software Tools area and he is an active member of the Automated Software Engineering and Visual Languages research communities. John is a Fellow of the Royal Society of New Zealand and a Member of the Ako Aotearoa Academy of Tertiary Teaching Excellence.

Yun Yang received his PhD degree from the University of Queensland, Australia, in 1992. He is a full professor at Swinburne University of Technology. His research interests include software engineering, cloud computing, workflow systems, and service-oriented computing.