IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 19, 2022, accepted May 12, 2022, date of publication May 17, 2022, date of current version June 6, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175850

A Game-Theoretic Approach for Increasing Resource Utilization in Edge Computing Enabled Internet of Things

SUMIT KUMAR[1], RUCHIR GUPTA[2], (Senior Member, IEEE), K. LAKSHMANAN[1], AND VIPIN MAURYA[1]

[1]Department of Computer Science and Engineering, Indian Institute of Technology (BHU) at Varanasi, Varanasi, Uttar Pradesh 221005, India
[2]School of Engineering, Jawaharlal Nehru University (JNU), Delhi 110067, India

Corresponding author: Ruchir Gupta (ruchirgupta@jnu.ac.in)

This work was supported in part by Nokia Solutions and Networks, Bangalore, India.

ABSTRACT *Edge computing* is a new paradigm that reduces latency and saves bandwidth by deploying edge servers in different geographic locations. This technology plays a crucial role in the rapidly growing app market for IoT devices as app vendors can hire computing resources on edge servers to serve their app users. An effective allocation of edge computing resources to different apps is needed to maximize resource utilization and serve the most app users at the lowest cost. We refer to this as an Edge Resource Allocation (ERA) problem. In this paper, we propose an Edge Resource Allocation Game (ERAGame), a game-theoretic approach that formulates the ERA problem by appropriately pricing the multi-tenant edge servers. The proposed approach gives a Pure Nash Equilibrium (PNE) solution to the ERA problem. For this, we design an ERA algorithm using ERAGame under which the system converges to PNE. For fast convergence to PNE, the edge servers are partitioned into different groups, enabling the ERA algorithm to run in parallel on all edge servers within each group. We prove that ERAGame is a potential game that guarantees at least one PNE under the ERA algorithm. We evaluate that the price of stability of ERAGame is at most $O(log\ n)$. The performance of the proposed algorithm is examined through simulation.

INDEX TERMS Edge computing, Internet of Things, resource allocation, app vendors, game theory, Nash equilibrium.

I. INTRODUCTION

Over the past decade, the world has seen exponential growth of the Internet of Things (IoT). These things integrate the cyber world with the physical world by sensing and collecting data from the surrounding environment and transmitting it to other devices over the Internet [1]. The rapid growth of IoT fueled the development of advanced apps. Due to limited resources, many resource-hungry IoT apps cannot process sensory data on IoT devices. To address this issue, IoT app users can offload complex computational tasks to the cloud [3], as shown in Fig. 1. In recent years, apps that require low latency have emerged, e.g., interactive gaming [4], natural language processing [5], face recognition [6], etc. However, due to unpredictable core network latency and expensive bandwidth, the cloud often fails to meet the stringent requirements of such latency-sensitive apps [7]. Edge computing technology is proposed to address these challenges [8]. This technology brings computing capability near the needed areas, saving energy and bandwidth while reducing latency.

In edge computing, servers are termed edge servers and deployed at various locations (generally near the cellular base stations) by the different service providers [9], e.g., Telstra, AT&T, etc., as shown in Fig. 1. These edge servers offer computing resources, e.g., memory, CPU, storage, etc. App vendors hire computing resources on the edge servers to serve their app users, deploy app-related services and software on edge servers, and assign app users to edge servers to offload computational tasks [10]. In recent years, the research on computation offloading in edge computing has attracted considerable attention from researchers [11]–[14]. Presently, research on resource allocation, energy savings, and latency reduction from the perspective of IoT devices or edge

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

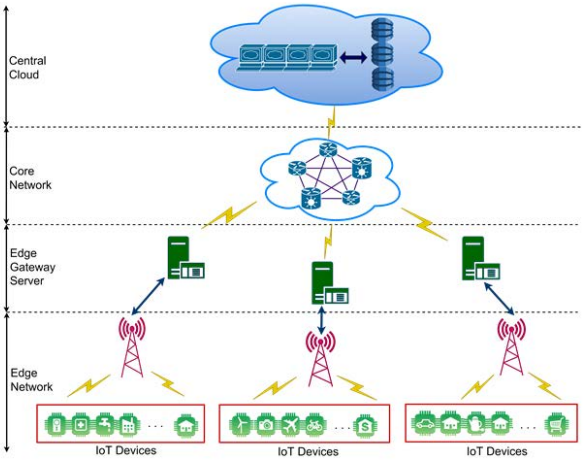


FIGURE 1. An example of edge and cloud computing enabled IoT.

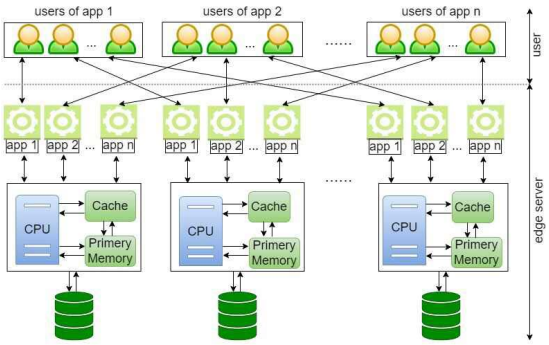


FIGURE 2. Multi-tenant architecture of edge servers.

computing infrastructure is being conducted [15]–[17]. This paper investigates the resource allocation on multi-tenant edge servers from the perspective of IoT app vendors. Multi-tenant architecture, commonly referred to as multi-tenancy, is a software architecture in which app instances of different applications run on a single physical machine where every single app instance serves multiple users, as shown in Fig. 2.

Generally, each edge server covers some geographical area, as shown in Fig. 3. The coverage areas of adjacent edge servers may overlap either to avoid blank sites or because of the availability of multiple edge servers from different infrastructure providers in any given geographic location [18]. Any edge server's computing resources can be allocated to various apps [19]. An IoT device in the overlapping region can connect to any edge server covering it (proximity constraint) with adequate computing resources (capacity constraint) [20]. Due to the capacity and proximity constraint, an inappropriate resource allocation to apps may result in many app users not being assigned to any edge server [20]. Therefore, app vendors must hire the resources of edge servers in a manner that maximizes the app users assigned to edge servers. App vendor pays for the computing resources of the edge servers hired from edge infrastructure providers [19], [21]. Thus, the other objective of each app vendor is to hire the optimal resources of the edge servers to reduce the cost. This can be accomplished by maximizing resource utilization and minimizing the required edge servers for app users. We refer to this problem of allocating resources to different app vendors (services) as the Edge Resource Allocation (ERA) problem.

Finding a centralized solution for the problem of edge servers' resource allocation to app vendors is an NP-hard problem [20], [21]. To address this issue, the researchers proposed heuristic approaches that find the sub-optimal solutions [21]–[23]. However, handling proximity and capacity constraints for multiple app vendors with a large number of users become a tedious task as the heuristic approach does not provide a desirable solution [24]. For better scalability, researchers proposed distributed approaches to solve the edge servers' resource allocation problem [18]–[20], [24]. Papers [19], [20] proposed the game-theoretic-based distributed approaches. In [19] and [20], each app user and app vendor can pursue individual interests. A game-theoretic approach gives the solution as a Pure Nash Equilibrium (PNE) where no app vendor or app user as a player can do better by unilaterally deviating from the solution.

In this paper, we propose a game-theoretic approach to solve the ERA problem. There are three primary motivations for adopting a game-theoretic approach [19], [20]: 1) It allocates the resources to different apps in a distributed manner by giving decision-making capability to each app vendor, which reduces the complexity of finding a solution for the ERA problem. 2) The app vendors can pursue different interests according to their needs in different computing capacity dimensions such as storage, bandwidth, and computational units. 3) A game-theoretic approach scales well with the size of the problem, e.g., the number of app users, number of services, number of edge servers, etc., as it works in a distributed manner.

We maximize resource utilization by effectively leveraging the multi-tenant edge servers, reducing the required edge servers for each app vendor. The proposed solution to the ERA problem minimizes the cost of each app vendor as it minimizes the required edge servers. Furthermore, this solution increases the overall assigned app users to given resources as it maximizes resource utilization. The key to maximizing resource utilization of any edge server is increasing the number of users using the same app on the edge server. With more users of the same service on each multi-tenant edge server, edge servers can be more effectively utilized since users can share global data variables, program code, cache contents, and other resources on a wide scale. As a result, this resource allocation also reduces service response time and latency.

The main contributions of this paper are as follows:

- An app vendor's cost for resource allocation on edge servers is defined by the number of edge servers used by the vendor and their resource utilization. The ERA problem is then modeled as a constrained optimization problem.

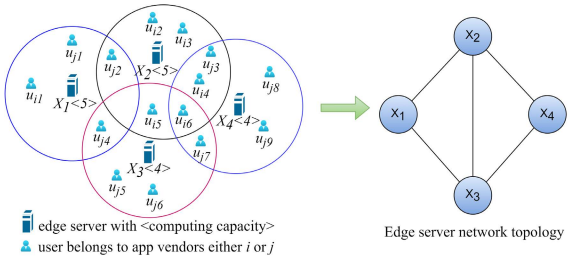


FIGURE 3. Edge servers' deployment with app users, and edge servers network topology.

- The constrained optimization problem is then formulated as an Edge Resource Allocation Game (ERAGame) to find a distributed solution, with app vendors acting as players. The preference of each app vendor is to acquire an optimal bundle of resources that serves the maximum app users on edge servers.
- A distributed Edge Resource Allocation (ERA) algorithm is proposed, under which all the app vendors collectively reach a PNE in the ERAGame.
- For fast convergence to PNE, the edge servers are partitioned into a few groups. These partitions allow the ERA algorithm to run parallel on each edge server within each group.
- The ESSGame is proven to be a potential game that accepts at least one PNE using the ERA algorithm.
- The performance of the ERA algorithm is evaluated theoretically against the central optimum solution.
- The ERA algorithm is compared numerically against five representative approaches for solving the ERA problem: a Best-Fit baseline, a Greedy baseline, and three state-of-the-art approaches.

The rest of the paper is organized as follows. Section II discusses the work related to app user allocation for task offloading. Section III elaborates on the ERA problem along with the system model. Section IV formulates the ERA problem as an ERAGame and analyzes the game property. Section V presents the ERA algorithm to achieve the PNE. Sections VI and VII analyze the ERA algorithm's performance.

II. RELATED WORK

Edge computing is a natural extension of cloud computing in terms of infrastructure deployment and network topology, with a more geographically distributed architecture than cloud computing. This architecture brings the benefits and capabilities of cloud computing to IoT devices by deploying edge servers in various locations closer to them. In the edge computing environment, IoT app vendors install required applications and services on edge servers to provide the computation capacities to their IoT app users [26]. The IoT app users offload their computation tasks to these edge servers [27]. Over the last decade, extensive research has been conducted on computational task offloading from IoT devices to edge servers [27]–[29].

IoT app users must be assigned to edge servers for offloading tasks from IoT devices to edge servers. From the perspective of app vendors, properly allocating IoT app users to edge servers with some optimization objectives is a critical problem [19]–[21]. The authors in [31] prove that allocating app users to edge servers with proximity and capacity constraints is equivalent to the variable-sized vector bin packing problem. The studies in [20], [21] verify that the ERA problem is an NP-hard problem. Finding a solution to the ERA problem becomes more difficult because each app user requires a different amount of various types of resources on edge servers [20], [25], [30].

To solve this problem in a polynomial time, researchers explore different way to find the solution for the ERA problem. In [32], the authors propose a heuristic solution to allocate the users to edge servers from the perspective of edge server infrastructure providers. The study [33] propose the dynamic service placement framework that gives an approximate solution handling end-user mobility for cost-efficient edge computing. The edge infrastructure deals automatically with end-user to edge server allocation.

In [34], [35], the authors assume that edge servers' coverage areas do not intersect each other or each small geographical is covered by a single edge server while allocating the app users to edge servers. Practically, it is unlikely to happen that each edge server covers an area exclusively [36], [37]. The researchers in paper [38] examine the edge resource allocation scenario with users' mobility, requiring the reallocation of users among edge servers. Their proposed approach to user allocation seeks to reduce the number of reallocations while maximizing users. These studies do not optimize the requisite servers for app vendors, which is one of our objectives.

Qiang *et al.* [20] and Phu *et al.* [21] propose game-theoretic based algorithms and heuristic approaches, respectively, to solve the ERA problem from the app vendor's perspective. However, they see the problem in respect of single app vendors and single infrastructure providers. The other dimension to view the app user allocation problem is to allocate the edge servers' resources to different apps (services). In [19], the authors investigate the resource allocation for edge computing, where multiple app vendors compete for computing resources. One of the major drawbacks of studies [19], [20] is that these proposed approaches take exponential iteration to reach Nash equilibrium.

In this paper, we propose a game-theoretic approach to find a solution for the Edge Resource Allocation (ERA) problem. The proposed approach has four important differences from state-of-the-art techniques: 1)It takes into account multiple app vendors while solving the ERA problem. 2) It proposes the designs of the ERA problem in a manner that efficiently formulates the multi-tenancy functionality of edge servers to utilize them effectively. 3) It leads to an increase in the utilization of the multi-tenant edge server by increasing the number of users using the same app on the edge server. 4) In the proposed approach, the convergence process for

finding the solution runs parallelly on the edge servers. As a result, it solves the problem more quickly than other state-of-the-art approaches.

III. SYSTEM MODEL

To solve the ERA problem from the perspective of app vendors, we consider n app vendors $V = \{1, 2, \cdots, n\}$, and each app vendor i has i_ρ app users $U_i = \{u_{i_1}, u_{i_2}, u_{i_3}, \cdots u_{i_\rho}\}$ at different locations. The app vendors' job is to hire computing capacity from m edge servers $S = \{1, 2, \cdots, m\}$ at different locations to serve app users. Various edge infrastructure providers own these edge servers. This paper only studies resource utilization from the perspective of the computing capacities belonging to the edge servers. The total available computing capacity of an edge server x is denoted by $\sigma_x = (\sigma_x^d)$, where $d \in D = \{$cache, memory, CPU, storage, $\cdots\}$. Fig. 2 depicts the organization of these various computing capacities, including cache in the edge servers. We assume that all the computing resources are managed at the edge server's operating system level. Our proposed approach works for any employed application, and it is independent of how the operating system manages computing capacities.

For numerical evaluation of the proposed approach, we create the simulation environment by considering some assumptions [39], [40] as follows. 1) The app vendors deploy the computational intensive calculation-based apps on the edge servers. 2) The tasks offloaded to an edge server by app users are queued to be executed on the CPU using Round Robin scheduling. 3) Before beginning the execution of an offloaded task, the task's availability in the cache memory is checked. If any component of the task is not already in cache memory, it is loaded into the cache memory using Least Recently Used algorithm.

Similarly, our proposed approach can be applied in other settings, and the results will be consistent. We examine the ERA problem in quasi-static scenarios where app users do not change computing capacity requirements and locations while allocating edge servers' resources, similar to other papers [12], [20], [41]. The notations adopted in the paper are summarized in Table 1.

Definition 1 (Allocation Decision): Given an app vendor i, $U_i = \{u_{i_1}, \cdots u_{i_\rho}\}$, and $S = \{1, \cdots, m\}$, an allocation decision of app vendor i is denoted by a vector $a_i = (\beta_{i,1}, \beta_{i,2}, \beta_{i,3}, \cdots, \beta_{i,m})$, where any $\beta_{i,x} = (\beta_{i,x}^d), d \in D$ refers to the hired computing resources of an edge server x by i. The set of all possible allocation decisions is referred to as A_i.

A. PROXIMITY CONSTRAINT

An app vendor i can only hire the computing resources of an edge server x for its app user u_{i_k} only if x's range, denoted by $cover(x)$, covers app user u_{i_k}, as follows:

$$u_{i_k} \in cover(x), \quad \forall u_{i_k} \in U_i, \ \forall i \in V, \ \forall x \in S \tag{1}$$

This proximity constraint means that an app user can only be assigned to one of the edge servers that covers the app user's location.

B. CAPACITY CONSTRAINT

The capacity constraint of each resource type d of each edge server x renders as follows: $\sum_{i=1}^{n} \beta_{i,x}^d \leq \sigma_x^d, \forall x \in S, \forall d \in D$, where $\beta_{i,x}^d \geq 0, \forall d \in D, \forall i \in V, \forall x \in S$.

Definition 2 (Allocation Decision Profile): Given $V = \{1, \cdots, n\}$, $U = \{U_1, \cdots U_n\}$, and $S = \{1, \cdots, m\}$, an allocation decision profile is the $n \times m$ matrix (denoted by a), in which the element at ith row and xth column is $\beta_{i,x}$. In other words, an allocation decision profile $a = (a_1, a_2, a_3 \cdots a_n)$ is the collection of all app vendors allocation decisions, where $a \in A = A_1 \times A_2 \times \cdots \times A_n$.

In the context of any app vendor i, the allocation decision profile can be written as (a_i, a_{-i}) where a_{-i} is the list of other app vendors' selection decision except i. Terms a, $(a_1, a_2, \cdots a_n)$, and (a_i, a_{-i}) have been used interchangeably throughout the paper.

1) EDGE SERVER NETWORK TOPOLOGY

The deployed edge servers remain stationary over time, and each edge server provides the service within the circled coverage area, as shown in Fig. 3. Any two edge servers whose coverage areas overlap are called neighboring edge servers. The edge servers network topology can be represented as graph $G(S, E)$, as shown in Fig. 3, where S is a set of edge servers (vertices), and E is a set of edges among the neighboring edge servers. The app vendors can hire the resources of the edge servers and deploy their app-related service software on these edge servers. The providers of the edge infrastructure can provide the edge server network topology to app vendors, which enables app vendors to move their services from one edge server to another according to their requirements [42].

2) SERVER UTILIZATION MODEL

Multi-tenancy maximizes the utilization of the available resources on the cloud [43] as well as the edge server [20]. The experimental results illustrated in [44] show the CPU utilization of an edge server x with multi-tenant architecture, which can be approximated as follows:

$$Z_{cpu} = log_\gamma(P) \tag{2}$$

where $P(P > 1)$ is the number of users allocated to the CPU unit, and $\gamma(.9 < \gamma < 1)$ is calculated by the computational task size. If an app vendor i hires $\beta_{i,x}^{cpu}$ units of CPU on edge server x, the average utilization of each unit of CPU will be,

$$Z(\beta_{i,x}^{cpu}) = \frac{\sum_{\alpha=1}^{\beta_{i,x}^{cpu}} log_\gamma(P_{CPU_{\alpha},x}^i)}{\beta_{i,x}^{cpu}} \tag{3}$$

where $P_{CPU_{\alpha},x}^i$ is the number of app users of app vendor i allocated to α^{th} unit of CPU on edge server x. Generally, the edge server's CPU utilization increases with the number of

TABLE 1. Symbol table.

Notation	Description
i/x	an app vendor / an edge server
V	the set of app vendors
S	the set of edge servers
U_i	the set of app users of app vendor i
σ_x	the total computing resources on edge server x
$\beta_{i,x}$	the allocated resources of x to vendor i
$\beta^d_{i,x}$	the type d allocated resources of x to app vendor i
a_i	the allocation decision of app user i
A_i	the edge server set contains neighbors of user i (allocation decision set of i)
$a = (a_i, a_{-i})$ $= (a_1, \cdots, a_n)$	the allocation decision profile
$a_{-i} =$ $(a_1, ..., a_{i-1}, a_{i+1}, ..., a_n)$	the allocation decision profile except for app user i
$P^i_{d_\alpha,x}$	the list contains the users of app vendor i assigned to resource type d of server x
R^a_x	the list contains app vendors who hire resources of edge server x
$\nu_i(a)$	the number of edge server used by i
f_x	the fixed cost of edge server x
$B(.)$	the usage cost incurred by app vendor
$c_{i,x}(a)$	the cost incurred by i for resources of x at a
$c_i(a)$	total cost incurred by app vendor i
$C(a)$	the total cost of all app vendors
$\Phi_x(a)$	the potential of edge server x due to
$\Phi(a)$	the overall potential of the game at a

app users as resource sharing on a multi-tenant edge server increases. If the number of users assigned to these resources on an edge server goes beyond a threshold, the increase in CPU utilization slows down and converges. It happens as the resource sharing overhead is too much after the threshold point. At some point, a multi-tenant server outperforms multiple single-tenant servers combined in overall CPU usage, which is also confirmed by [43]. The memory and storage usage in a server with multi-tenant architecture follows similar patterns as CPU usage [20], [43]. This paper assumes that multi-tenant servers' other computing resources follow identical patterns as storage, memory, and CPU, as in [20]. Thus, utilization of resources $\beta^d_{i,x}$ of type d hired by app vendor i on multi-tenant edge server x can be determined as follows:

$$Z(\beta^d_{i,x}) = \frac{\sum_{\alpha=1}^{\beta^d_{i,x}} log_\gamma(P^i_{d_\alpha,x})}{\beta^d_{i,x}} \tag{4}$$

where $d \in D$ is the resource type, and γ_d is calculated by the computational task size. γ_d is specific to each type of computing resource as the computational task size impacts each resource type differently. We assume that the maximum size task offloaded by each app user of the same service is similar in size, as in [20]. Thus, γ_d is service-specific and does not vary during the allocation process.

3) APP VENDOR COST MODEL

The infrastructure providers rent out the computational resources based on a pricing model. The cost incurred by an app vendor to hire the computing resources will comprise two components: usage cost and fixed cost. In this paper, we use a cost model that determines the usage cost of computing resources based on their utilization. The pricing model based on resource utilization is called the pay-as-you-go pricing model used by different infrastructure providers, e.g., AWS, Azure, Salesforce, etc. The utilization of resources of an edge server hired by an app vendor can be determined from Eq. 4. If an app vendor i hires computing resources $\beta_{i,x} = (\beta^d_{i,x})$, $d \in D$ of edge server x, the usage cost, denoted by B, can be determined as follows:

$$B(\beta_{i,x}) = \sum_{d \in D} \delta_d \cdot Z(\beta^d_{i,x}) \cdot \beta^d_{i,x} \tag{5}$$

where δ_d is the application-specific priority assigned to resource type d indicates the significance of resource type d for service provided by the app vendor. The usage cost $B(\beta_{i,x})$ is a non-decreasing concave function as $Z_d(\beta^d_{i,x})$. Thus, the usage cost per user of app vendors decreases as the resource utilization increases. In other words, this cost function motivates the vendor to increase the number of users of the same app on an edge server to reduce the usage cost. The usage cost optimization increases the app users assigned to a given bundle of resources and minimizes the required edge servers per app.

The fixed cost, denoted by F, incurred by app vendor i can be determined as follows:

$$F(a, x) = \frac{f_x}{|R^a_x|} \tag{6}$$

where R^a_x is the number of app vendors who hire the computing resources on edge server x, and f_x is the cost of edge server x. Cost f_x may include cost of electricity consumption in the building, land, maintenance, depreciation of machine, managerial and administrative staff, etc. The optimization of fixed cost $F(a, x)$ incurred by each app vendor motivates the vendor to hire the computing resources on an edge server that provides services to users of more apps. As a result, this optimization increases the overall utilization of an edge server subject to capacity constraints.

The total cost incurred by each app vendor i for hiring resources on edge server x at selection decision profile a can be calculated as,

$$c_{i,x}(a) = \frac{f_x}{|R^a_x|} + B(\beta_{i,x}) \tag{7}$$

An app vendor i may hire computing resources on various edge servers and maintains a list $\nu_i(a)$ of these edge servers. Thus, the overall cost incurred by i for all edge servers in $\nu_i(a)$ at allocation decision profile a is,

$$c_i(a) = \sum_{x \in \nu_i(a)} c_{i,x}(a) = \sum_{x \in \nu_i(a)} \left(\frac{f_x}{|R^a_x|} + B(\beta_{i,x}) \right) \tag{8}$$

If the cost vector is $c(a) = (c_1(a), c_2(a), c_3(a), \cdots, c_n(a))$, where any $c_i(a)$ denotes the cost incurred by app vendor i at allocation decision profile a, the total cost of all the app vendors, denoted by $C(a)$, will be as,

$$C(a) = \sum_{i \in V} c_i(a) \tag{9}$$

4) OPTIMIZATION MODEL

In this paper, we model the ERA problem as a constrained optimization problem. Given app vendor set $V = \{1, 2, \cdots, n\}$ and edge server set $S = \{1, 2, \cdots, m\}$, the constrained optimization problem can be modeled as follows:

$$\min_{a \in A} \sum_{i \in V} c_i(a) \tag{10}$$

subject to:

$$u_{i_k} \in cover(x), \quad \forall u_{i_k} \in U_i, \ \forall i \in V, \ \forall x \in S \tag{11}$$

$$\sum_{i=1}^{n} \beta_{i,x}^{d} \leq \sigma_x^d, \quad \forall x \in S, \ \forall d \in D \tag{12}$$

$$\beta_{i,x}^{d} \geq 0, \quad \forall d \in D, \ \forall i \in V, \ \forall x \in S \tag{13}$$

Objective 10 minimizes the app vendors' total cost. Constraint 11 ensures that every app user u_{i_k} should be allocated to an edge server x only when x covers u_{i_k}. Constraints 12 and 13 ensures that the computing capacity hired by app vendors on an edge server x must not exceed of available computing capacities of x.

IV. EDGE RESOURCE ALLOCATION GAME

This section formulates the ERA problem as the Edge Resource Allocation Game (ERAGame). As aforementioned, the app vendors share edge servers' costs. Hence, one app vendor's allocation decision regarding hiring and abandoning the computing resources of an edge server affects the cost of other app vendors. Thus, app vendors have a strategic interaction as the cost incurred by an app vendor depends on its allocation decision as well as the other's allocation decisions. The strategic interaction can be modeled as an ERAGame, where each app vendor acts as a player. ERAGame aims to find an allocation decision profile $a^* = (a_1^*, \cdots, a_n^*), a^* \in A$ as a Pure Nash Equilibrium (PNE) in which the cost vector $c(a^*) = (c_1(a^*), \cdots, c_n(a^*))$ that includes the cost incurred by all the app vendors is stable optimal.

Definition 3 (Pure Nash Equilibrium): An allocation decision profile $a^* = (a_1^*, a_2^*, a_3^* \cdots a_n^*)$ is a PNE if no app vendor can reduce its cost by unilaterally deviating from its allocation decision, i.e,

$$c_i(a_i^*, a_{-i}^*) \leq c_i(a_i, a_{-i}^*) \text{ for all } a_i \in A_i, \quad i \in V \tag{14}$$

In ERAGame, each app vendor i's objective is to minimize cost $c_i(a)$ at every allocation decision profile a. In more elabrotely, given other app vendors' decisions a_{-i}, app vendor i would like to perform a suitable decision a_i to minimize $c_i(a)$ as:

$$\min_{a_i \in A_i} c_i(a_i, a_{-i}) \tag{15}$$

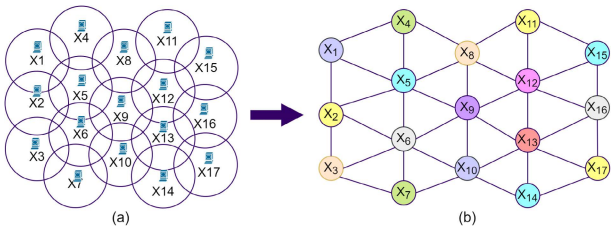


FIGURE 4. (a) Edge servers deployment, and (b) partition of edge servers into the groups.

ERAGame formulates the problem as a tuple $(V, \{A_i\}_{i \in V}, \{c_i\}_{i \in V})$, where V is a set of app vendors, A_i is i's finite set of allocation decisions subject to Eqs. 11-13, and c_i is the cost function that determine the cost of i's selection decision $a_i \in A_i$. In ERAGame, all the app vendors gradually move towards a PNE to reducing their costs. Whether ERAGame accepts at least one PNE is of critical significance to this study.

V. DISTRIBUTED RESOURCE ALLOCATION MECHANISM

This section proposes an Edge Resource Allocation (ERA) algorithm to find a Pure Nash Equilibrium (PNE) allocation decision profile. As app vendors have strategic interaction, each app vendor makes a resource allocation decision to respond to the other app vendors' decisions in ERAGame. Each allocation decision made by the app vendor is seen as an improvement in terms of cost-cutting and resource-utilization enhancement. Suppose app vendors make allocation decisions at the same time. In that case, the outcome of each app vendor allocation decision may not be regarded as an improvement since they affect each other's incurred costs. Hence, to ensure progress with each allocation decision, only one app vendor at a time can be allowed to make allocation decisions with respect to allocation decisions chosen by other app vendors. However, it may take a considerable amount of time to converge at PNE.

To achieve fast convergence, we first partition the edge server set into different groups and feed these groups to the ERA algorithm as an input. The edge server groups must be created in such a way that resource allocation decisions made on one edge server in a group do not directly impact decisions made on other edge servers within that group. This kind of grouping will allow the ERA algorithm to run parallel on all the edge servers within each group, leading to fast convergence to PNE. However, the existence and quality of PNE in ERAGame are irrespective of the process of grouping and the number of edge server groups.

A. EDGE SERVER GROUPING (SYSTEM PREPROCESSING)

1) DESIRE PROPERTIES OF GROUPS

In each group, every pair of edge servers must be more than two edges apart (distance between nodes). For example, edge servers X_4 and X_7, shown in Fig. 4b, can be included in the same group since the shortest path length

between X_4 and X_7 is more than two. Such distance among edge servers allows app vendors to allocate and deallocate edge server resources without directly influencing the allocation and deallocation of resources on other edge servers in the group. This can be illustrated by an example using Fig. 4 as follows. The coverage area of edge servers X_4 and X_7 overlap with $\{X_1, X_5, X_8\}$ and $\{X_3, X_6, X_{10}\}$, respectively, and $\{X_1, X_5, X_8\} \cap \{X_3, X_6, X_{10}\} = \emptyset$. The edge servers in $\{X_1, X_5, X_8\}$ and $\{X_3, X_6, X_{10}\}$ are neighboring edge servers of X_4 and X_7, respectively. If constraints in Eqs. 11-13 are met, the users of an app assigned to edge server X_4 who are also present in the overlapping coverage areas of X_4 and $\{X_1, X_5, X_8\}$ can be assigned from X_4 to $\{X_1, X_5, X_8\}$. Users in overlapping coverage areas of X_4 and X_1 can move from X_4 to X_1; users in overlapping coverage areas of X_4 and X_5 can move from X_4 to X_5; users in overlapping coverage areas of X_4 and X_8 can move from X_4 to X_8. App users who are not within the overlapping coverage area cannot be assigned to edge servers in $\{X_1, X_5, X_8\}$. Thus, any vendor who has hired resources on X_4 for its app users in the overlapping area can hire resources on neighboring servers in $\{X_1, X_5, X_8\}$ rather than X_4. Similarly, an app vendor with resources on X_7 can redirect app users to an edge server in $\{X_3, X_6, X_{10}\}$ rather than X_7. As the intersection of $\{X_1, X_5, X_8\}$ and $\{X_3, X_6, X_{10}\}$ is empty, resource deallocation from edge servers X_4 and X_7 to other edge servers can be done in parallel without interfering with each other.

Thus, each group has the property that if an app vendor relocates users from one edge server, the decision does not directly influence the relocation decision of users from other servers within that group from the perspective of cost and capacity. This property allows the ERA algorithm to be executed in parallel on all edge servers belonging to a group, reducing the time complexity. Thus, m edge server set can be divided into different groups as group set $\Pi = \{\Pi_0, \Pi_2, \cdots, \Pi_{T-1}\}$, where any $\Pi_l \in \Pi$ is a group of edge servers, and T is the number of groups. These groups have the properties as follows:

- $distance(x_1, x_2) > 2$, for every $x_1, x_2 \in \Pi_l$, for all $\Pi_l \in \Pi$, where $distance(x_1, x_2)$ represents the distance between x_1 and x_2.
- $\Pi_{l1} \cap \Pi_{l2} = \emptyset$, for every $\Pi_{l1}, \Pi_{l2} \in \Pi$

2) GROUPS FORMATION

The edge server set can be partitioned into a few groups using the network topology of edge servers provided by the edge computing infrastructure providers. These groups can be formed using *distance-2 graph coloring*. If the distance between two edge servers is more than two, the distance-2 coloring of the graph assigns the same integer number to both edge servers. Edge servers with identical integer numbers are grouped in the same group, and the total number of integers used in the distance-2 graph coloring represents the number of groups. More elaborately, if T is the number of colors and each distinct integer number $l \in \{0, 1, 2, \cdots, T-1\}$ refers to a distinct color, the distance-2 coloring of graph $G(S, E)$ is a mapping from edge server set S to $\{0, 1, 2, \cdots, T-1\}$ such that each pair of edge servers having a distance at most 2 receive distinct integer numbers from set $\{0, 1, \cdots, T-1\}$. The edge servers receiving the same integer number $l \in \{0, 1, \cdots, T-1\}$ belong to group Π_l. For example (Fig. 4b), edge servers $\{X_2, X_{11}, X_{17}\}$ depicted by yellow color form a group.

To partition the edge server into the groups, we use the heuristic-based *distance-2 graph coloring* algorithm [45]. The complexity of finding such groups is $O(m \times |E|)$, where m is the number of edge servers, and E is a set of edges among the edge servers. The group formation process takes place once because the edge servers are static entities. If the infrastructure provider extends the geographical service area by increasing the number of edge servers, new edge servers can be included in the existing groups based on their distance property.

B. EDGE RESOURCE ALLOCATION ALGORITHM

Given $V = \{1, \cdots, n\}$, $\{U_i, \cdots, U_n\}$ and $S = \{1, \cdots, m\}$, ERAGame applies an iterative process for convergence to the PNE, as shown in Algorithm 1. In each iteration, app vendors try to minimize their costs by following the ERAGame rules while gradually moving towards the PNE. The process begins with an arbitrary allocation decision profile a, and an app vendor list R_x^a is created for every edge server (Lines 1-2). The process iterations are repeated for every group until the system reaches PNE. Each iteration is represented by $t(t = 1, 2, 3 \cdots)$, and the allocation decision profile in any t is denoted by $a(t)$.

In every iteration, each edge server sends a message containing $< SI_x, R_x^{a(t)}, \lambda_x^{a(t)} >$ to neighboring edge servers (Line 4), where SI_x is the identity of edge server x, $R_x^{a(t)}$ is the list of app vendors, and λ_x^a is the available resources of edge server x at allocation decision profile $a(t)$. This message is transmitted in a decentralized manner which needs messaging synchronization [4]. Next, a group Π_l is selected, where $l \equiv t \pmod T$ (Line 5). The computation in each iteration of the allocation process (Lines 6-20) is performed by individual app vendors on each edge server $y \in \Pi_l$ in parallel. After selecting group Π_l, each app vendor i with resources on edge servers in group Π_l can participate in the current iteration to minimize its cost (Line 6-7). Then each participant app vendor i on each edge server $x \in \Pi_l$ finds the set of optimal alternative edge servers NEG_x of x in terms of cost. This optimal set must satisfy the proximity constraints Eq. 11 and capacity constraints Eqs. 12-13. This means that the edge servers in NEG_x can give the services to i's app users currently getting the services from x (Line 8). Next, if the app vendor i's possible cost on edge servers in NEG_x is less than the cost incurred on x, the app vendor i requests resource allocation on the edge servers in NEG_x (Lines 9-13). The resources are allocated to any app vendor on every edge server $y \in NEG_x$ according to (Lines 14-18). If any edge server receives multiple resource allocation requests from

Algorithm 1: Edge Resource Allocation

Input: $V, \{U_i, \cdots, U_n\}, G(S, E), \Pi$
Output: Allocation Decision Profile a
Result: A Pure Nash Equilibrium

1 **Initialize** $a \leftarrow (a_1, a_2, a_3, \cdots a_n)$ to be an arbitrary allocation decision profile
2 At every edge server $x \in S$ do $R_x^a \leftarrow \{i \mid i \text{ is an app vendor hired resources on } x \text{ at } a\}$
3 **repeat**
4 each edge server $x \in S$ send a message containing $< SI_x, R_x^{a(t)}, \lambda_x^{a(t)} >$ to its neighbouring edge servers
5 selection of edge server group Π_l for iteration t
6 **for** *each edge server* $x \in \Pi_l$ **do**
7 **for** *each app vendor* $i \in R_x^{a(t)}$ **do**
8 i finds each set of alternative edge servers NEG_x of x, subject to constraints 11, 12, and 13
9 **for** *each set of edge servers* NEG_x **do**
10 i calculates the possible cost of the resource allocation on each edge server in NEG_x as in Eq. 7
11 i finds an edge server set NEG_x^{min} such that $\min_{NEG_x}(\sum_{y \in NEG_x} c_{i,y}(a(t)))$
12 **if** $\sum_{y \in NEG_x^{min}} c_{i,y}(a(t)) < c_{i,x}(a(t))$ **then**
13 app vendor i request for resource allocation on each $y \in NEG_x^{min}$
14 **for** *each* $y \in NEG_x^{min}$ **do**
15 **if** *y receive request from multiple vendors* **then**
16 edge server y accepts the requests that best fit the available resources
17 **else**
18 accept the request of app vendor i
19 **if** *app vendor i's request accepted* **then**
20 deallocate the resources from x
21 **until** *no more allocation decision updates needed*;

app vendors, it accepts the requests that can best fit the available resources $\lambda_x^{a(t)}$. In this case, an edge server y can accept the request of app vendor i if it is included in the best fit. If app vendor i's request is accepted by all edge servers in $y \in NEG_x$, the allocated resources on edge server x for the services provided by app vendor i are deallocated (Lines 19-20). These iterations repeat for each group $\Pi_l \in \Pi$ until the app vendors stop updating their selection decisions (Line 21). App vendors' individual selection decisions comprise the final allocation decision profile that will be a solution as the PNE.

VI. THEORETICAL ANALYSIS

A. GAME PROPERTY

In the current configuration of the ERAGame, all app vendors try to minimize their costs by employing the ERA algorithm. As a result, the ERA algorithm behaves similarly to a gradient descent process to identify a PNE by exploring all possibilities. To investigate the existence of PNE in the ERAGame, we need a function defined on the allocation decision profile that observes the convergence process of the ERA algorithm. The overall cost of all app vendors can not be used to observe the convergence process because it is inconsistent during the convergence process. This is because when an app vendor changes its allocation decision to reduce its cost, this impacts the distribution of edge servers' fixed costs among other app vendors. As a result, the pattern of change in the overall cost of the system is not steady. We need a consistent function that should always decrease with any change in allocation decision made by any app vendor using the ERA algorithm. The key is to find a potential function and to establish that ERAGame is a potential game defined as follows:

Definition 4 (Potential Game): A game is a potential game if there is a potential function $\Phi(a)$ such that,

$$c_i(a_i', a_{-i}) \leq c_i(a_i, a_{-i}) \Rightarrow \Phi(a_i', a_{-i}) \leq \Phi(a_i, a_{-i}) \quad \text{for any } i \in V, \quad a_i, a_i' \in A_i \text{ and } a_{-i} \in \prod_{l \neq i} A_l \tag{16}$$

We can determine the potential function $\Phi(a)$ value of the game at any allocation decision profile a by calculating the potential of each edge server. The potential of an edge server can be determined as follows. Suppose only one app vendor i hires computing resources $\beta_{i,x}$ of an edge server x. In that case, the edge server's potential is equal to cost incurred by app vendor for hiring resources on edge server x. Therefore, the edge server's potential is $f_x + B(\beta_{i,x})$. When another app vendor j hires the computing resources of edge server x, x's potential will be as the previous potential of x + the cost incurred by app vendor j for hiring the resources on x. The cost incurred by app vendor j is $\frac{f_x}{2} + B(\beta_{j,x})$. Hence, the potential of edge server x is increased to $f_x + \frac{f_x}{2} + B(\beta_{i,x}) + B(\beta_{j,x})$ and so on. Thus, the potential of an edge server x with app vendors R_x^a at an allocation decision profile a is,

$$\Phi_x(a) = \sum_{k=1}^{|R_x^a|} (\frac{f_x}{k}) + \sum_{i \in R_x^a} B(\beta_{i,x}) \tag{17}$$

The overall potential $\Phi(a)$ of ERAGame can be calculated by summing the potential of all edge servers as,

$$\begin{aligned} \Phi(a) &= \sum_{x \in S} \left(\sum_{k=1}^{|R_x^a|} (\frac{f_x}{k}) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \\ &= \sum_{x \in S} \left(f_x \cdot H(|R_x^a|) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \end{aligned} \tag{18}$$

where $H(|R_x^a|) = \left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{|R_x^a|}\right)$.

Theorem 1: The ERAGame with the potential function $\Phi(a)$ is a potential game.

Proof: Suppose an app vendor i changes its allocation decision from a_i to a_i' to minimize its cost and the allocation decision profile changes from a to a'. The app vendor sees this change as an improvement. Consequently, the change in i's cost will be negative as,

$$\Delta c_i = c_i(a') - c_i(a) < 0 \tag{19}$$

Due to deviation, the list of edge servers on which app vendor i has computational resources is updated from $\nu_i(a)$ to $\nu_i(a')$, where $\nu_i(a') \setminus \nu_i(a)$ and $\nu_i(a) \setminus \nu_i(a')$ contain all the edge servers added and left by i, respectively. Only edge servers in $\nu_i(a') \setminus \nu_i(a)$ and $\nu_i(a) \setminus \nu_i(a')$ impact the cost of app vendor i. The change in app vendor i's cost from Eqs. 8 and 19 can be elaborated as,

$$\begin{aligned}\Delta c_i &= \sum_{x \in \nu_i(a') \setminus \nu_i(a)} c_{i,x}(a) - \sum_{y \in \nu_i(a) \setminus \nu_i(a')} c_{i,y}(a') \\ &= \sum_{x \in \nu_i(a') \setminus \nu_i(a)} \left(\frac{f_x}{|R_x^a| + 1} + B(\beta_{i,x}) \right) \\ &\quad - \sum_{y \in \nu_i(a) \setminus \nu_i(a')} \left(\frac{f_y}{|R_y^a|} + B(\beta_{i,y}) \right) < 0 \end{aligned} \tag{20}$$

where R_x^a or R_y^a is a list of app vendors that hire computing resources of x or y, respectively, before app vendor i deviates. The change in Potential function value is,

$$\Delta\Phi = \Phi_{Inc} - \Phi_{Dec} \tag{21}$$

The app vendor i hires the computing resources on the edge servers in $\nu_i(a') \setminus \nu_i(a)$. As a result, the potential of these edge servers will increase. Therefore, from Eqs. 17 and 18, the increase in potential is $\Phi_{Inc} = \sum_{x \in \nu_i(a') \setminus \nu_i(a)} \left(\frac{f_x}{|R_x^a|+1} + B(\beta_{i,x}) \right)$. On the other hand, i abandons the computing resources of edge servers in $\nu_i(a) \setminus \nu_i(a')$. Hence, from Eqs. 17 and 18, the value of Φ decreases by $\Phi_{Dec} = \sum_{y \in \nu_i(a) \setminus \nu_i(a')} \left(\frac{f_y}{|R_y^a|} + B(\beta_{i,y}) \right)$. Thus, from Eqs. 20 and 21,

$$\Delta\Phi = \Delta c_i < 0 \tag{22}$$

From Eq. 22, if any app vendor i changes its allocation decision to minimize its cost, $c_i(a') \leq c_i(a)$ implies $\Phi(a') \leq \Phi(a)$. Hence, ERAGame is a potential game. □

Theorem 1 also proves that the potential function's value decreases with each change in the allocation decision made by any app vendor as an improvement. In the ERA algorithm, app vendors make the allocation decision to minimize their costs. Therefore, the value of the potential function will decrease with every iteration of the ERA algorithm.

Theorem 2: In ERAGame, the system converges to at least one PNE.

Proof: The existence of PNE in the ERAGame can be demonstrated using the following two reasons:

1) The ERAGame contains a finite number of allocation decision profiles, and the potential function $\Phi(a)$ is defined on them. Consequently, the outcomes of $\Phi(a)$ are also finite.
2) According to Theorem 1, the value of the potential function $\Phi(a)$ declines monotonically with each change in the app vendors' allocation decisions under the ERA algorithm; thus, no cycle is feasible in the ERAGame under the ERA algorithm.

Both arguments imply that the ERA algorithm will eventually halt. The halting point signifies that no app vendor benefits by deviating from the current allocation decision, which is essential for PNE existence. Hence, there exist at least one PNE in the ERAGame. □

B. CONVERGENCE ANALYSIS

This subsection analyzes how many iterations of the ERA algorithm are required to converge at PNE by following the trajectory of the convergence process. The potential function is used to observe the convergence process of the ERA algorithm, as aforementioned. In each iteration of the ERA algorithm, the app vendors make the allocation decision for their cost reduction and collectively generate an allocation decision profile. This process continues until the system reaches the PNE. The app vendors reduce their costs in every iteration of the ERA algorithm, so the potential $\Phi(a)$ value would monotonically decrease in every iteration, as demonstrated in Theorem 1. This implies that the allocation decision profile is not repeated during the execution of the ERA algorithm employed in the finite ERAGame. Theorem 2 proved that the system reaches the PNE in a finite number of iterations of the ERA algorithm. Moreover, the edge server set is partitioned into T groups. In each iteration of the ERA algorithm, app vendors can make allocation decisions in parallel on every edge server in any group $\Pi_l \subseteq \Pi$. From these conclusions, it follows that the number of iterations in the ERA algorithm to reach Nash equilibrium will never exceed $\max_{q_{i,x}}(T \times q_{i,x})$, where $q_{i,x}$ is the number of all possible alternates in place of an edge server x for app vendor i.

C. PRICE OF STABILITY

This subsection evaluates the bound on the quality of the PNE solution obtained using the ERA algorithm in order to meet app vendors' optimization objectives, as discussed in Section III. The best PNE has the lowest overall cost of app vendors. In general, the Price of Stability (PoS), which is the ratio of the overall cost of app vendors at best PNE to the centralized optimum, demonstrates the quality of the PNE. If allocation decision profiles a^* and a^o are the best PNE solution and centralized solution, respectively, the PoS is,

$$PoS(a) = \frac{C(a^*)}{C(a^o)} \tag{23}$$

where $C(a^*)$ and $C(a^o)$ are overall costs incurred by app vendors at allocation decision profiles a^* and a^o, respectively. Since the potential function plays a vital role in the estimation

of PoS, we first establish a relationship between the potential function and the total cost of app vendors.

Lemma 1: In ERAGame, for any allocation decision profile, $C(a) \leq \Phi(a) \leq H(n) \cdot C(a)$, where $C(a)$ is the total cost incurred by n app vendors.

Proof: From Eq. 10, n app vendors' total cost $C(a)$ is,

$$C(a) = \sum_{x \in S} \left(f_x + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \tag{24}$$

As defined in section VI-A, $H(|R_x^a|) \geq 1$ for any $|R_x^a| \geq 1$. Thus,

$$C(a) \leq \sum_{x \in S} \left(f_x \cdot H(|R_x^a|) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) = \Phi(a) \tag{25}$$

Both $H(.)$ and $B(.)$ are non-negative and non-decreasing functions, and $H(n) \geq H(R_x^a)$ as $n \geq R_x^a$. So,

$$\Phi(a) \leq H(n) \cdot \sum_{x \in S_a} (f_x + \sum_{i \in R_x^a} B(\beta_{i,x})) = H(n) \cdot C(a) \tag{26}$$

Hence, from Eqs. 25 and 26, $C(a) \leq \Phi(a) \leq H(n) \cdot C(a)$. □

Theorem 3: In ERAGame, the price of stability is at most H(n).

Proof: Suppose the allocation decision profile a^o is a central enforced optimum solution and $C(a^o)$ is the cost at a^o. We assume that app vendors begin with a^o as the initial allocation decision profile and changing allocation decisions until the system reaches a PNE a^*. As stated in the Theorem 1, $\Phi(a)$ will decrease with every improvement of app vendors in this process, so,

$$\Phi(a^*) \leq \Phi(a^o) \tag{27}$$

From the first inequality of lemma 1,

$$C(a^*) \leq \Phi(a^*) \tag{28}$$

From inequalities in Eqs. 27 and 28,

$$C(a^*) \leq \Phi(a^o) \tag{29}$$

The second inequality of lemma 1 says,

$$\Phi(a^o) \leq H(n) \cdot C(a^o) \tag{30}$$

Hence, the result infers from the inequalities in Eq. 29 and 30, $C(a^*) \leq H(n) \cdot C(a^o) \Rightarrow \frac{C(a^*)}{C(a^o)} \leq H(n)$ □

In ERAGame, there is a PNE solution, for which the total maximum cost of all app vendors will not exceed the $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = O(\log n)$ factor of the cost at the optimal centralized solution.

VII. NUMERICAL EVALUATION

This section evaluates the ERA algorithm's performance by conducting a set of extensive experiments on the different combinations of edge servers, app vendors and app users.

TABLE 2. Experimental settings.

	Number of app vendors	Number of users per app	Number of edge servers
set-1	25	100,...,1000	50
set-2	25	500	10,...,100
set-3	5,...,50	500	50

A. PERFORMANCE BENCHMARK

The proposed ERA algorithm is evaluated against five representative approaches, namely a Best-Fit baseline, a Greedy baseline, and three state-of-the-art approaches for solving the ERA problem. These baseline and state-of-the-art approaches are as follows:

- Best-Fit: This approach allocates the edge server resources to an app vendor in such a way that the remaining resources are as small as possible, subject to proximity constraints.
- Greedy: This approach allocates the resources to any app vendor on an edge server with the maximum available resources, subject to proximity constraints.
- PRDS [19]: The resource allocation problem for various services with budget limitations was solved using a Game-Theoretic approach. Its goal is to allocate the optimal bundle of resources to each service.
- TPDS [20]: This research proposed a game-theoretic approach for allocating app users to edge servers. Its goal is to increase the number of app users assigned to the edge servers while lowering the overall cost.
- MCFH [21]: This solution uses a heuristic to solve the app user allocation problem by first allocating the highest capacity edge server. Its objective is to maximize the number of app users on edge servers while reducing the number of edge servers required.

PRDS, TPDS, and MCFH solves the same resource allocation problem as studied in this paper. We select the PRDS, TPDS, and MCFH algorithms as the state-of-the-art benchmark. All the experiments are written in Python 3.5. These experiments are conducted on Windows 10 OS system with Intel CORE i7-7700U CPU 3.60 GHz and 8 GB RAM.

B. SIMULATION SETTINGS

In this area, the works generally use the custom-built simulator in different languages. In the same way, as other papers [19]–[20] in this area used to develop the simulation environment, we created our environment in Python 3.5 to implement all of the experiments. We have set up a 20 km × 20 km rectangular space to evaluate the algorithm. The base stations are spaced 600 meters apart throughout this rectangular region. For each simulation, we deploy 100 edge servers at different base stations at random. Edge servers' coverage radius is uniformly distributed over [750 m, 1500 m]. The available resources on each edge server are produced at random using a normal distribution $N(\mu, \sigma^2)$, with $\mu = 50$ being the average amount of each resource type d, and $\sigma = 10$ representing the standard deviation. Each type's

generated number of computing resource units is rounded off to a positive integer number. Under the normal distribution, a negative amount of any resource type can be generated; hence any negative value is rounded to 1. In our experiments, we consider resource type set $D = \{bandwidth, Storage, Memory, Cache, CPU\}$. The app users' offloaded work is generated by uniform distribution over [10KB, 60KB]. The number of app users R_x^a and parameter γ_d determine the resource consumption $Z_d(x)$ of type d on server x. The average task size, denoted by Avg_Size, offloaded to edge server determines the value of γ_d. For each resource type d in our experiment, the value of γ_d is set to $\{(10KB \leq Avg\_Size < 20\ KB, \gamma_d = 0.91), (21KB \leq Avg\_Size < 30\ KB, \gamma_d = 0.92), (31KB \leq Avg\_Size < 40\ KB, \gamma_d = 0.93), (41KB \leq Avg\_Size < 50\ KB, \gamma_d = 0.94), (50KB \leq Avg\_Size < 60\ KB, \gamma_d = 0.95)\}$. Priority δ_d is set to 0.25 for each resource type d. Any app user's resource requirements are determined by the size of the offloaded task. We presume that a task with less than 30 KB requires one unit of each resource type, whereas a task with a size of more than 30 KB requires two. The fixed cost of each edge server is generated by uniform distribution over [50], [75]. We create the simulation setting to calculate the response time as in the paper [39], [40]. We assume a cache memory of 128 kb in size, with each block being 4kb in size. Each app user's computational task is divided into pages of 4 kb in size. We assume that for each app, the first 50% of total pages of computational tasks offloaded by any app user is shared with other app users of the same app. The Least Recently Used (LRU) page replacement technique swaps the pages from the cache. The Round Robin scheduling algorithm is used for CPU allocation with a time quantum of 10 ms. The response time for an app user is calculated as the amount of waiting time in the queue for first-time CPU allocation and the time required to cache all of its computational task's pages. We assume that every 1 kb of the offloaded task by app users needs a CPU burst of 1 ms. The cache access (page hit) and primary memory access time (page miss) are assumed to be 20 nanoseconds and 800 nanoseconds. The computational tasks by each app user are generated at some intervals. These intervals are selected by uniform distribution over [2 seconds, 10 seconds].

Experiments are run on the data sets by changing various parameters such as app vendors, app users, and edge servers. The data sets are summarized in Table 2. Each experiment is repeated 50 times, and the results are averaged. It allows neutralizing extreme cases like sparse or dense server/app user distributions. The error bar with a 95 percent confidence interval is depicted by the vertical lines in the result figures of each experiment.

C. PERFORMANCE COMPARISON

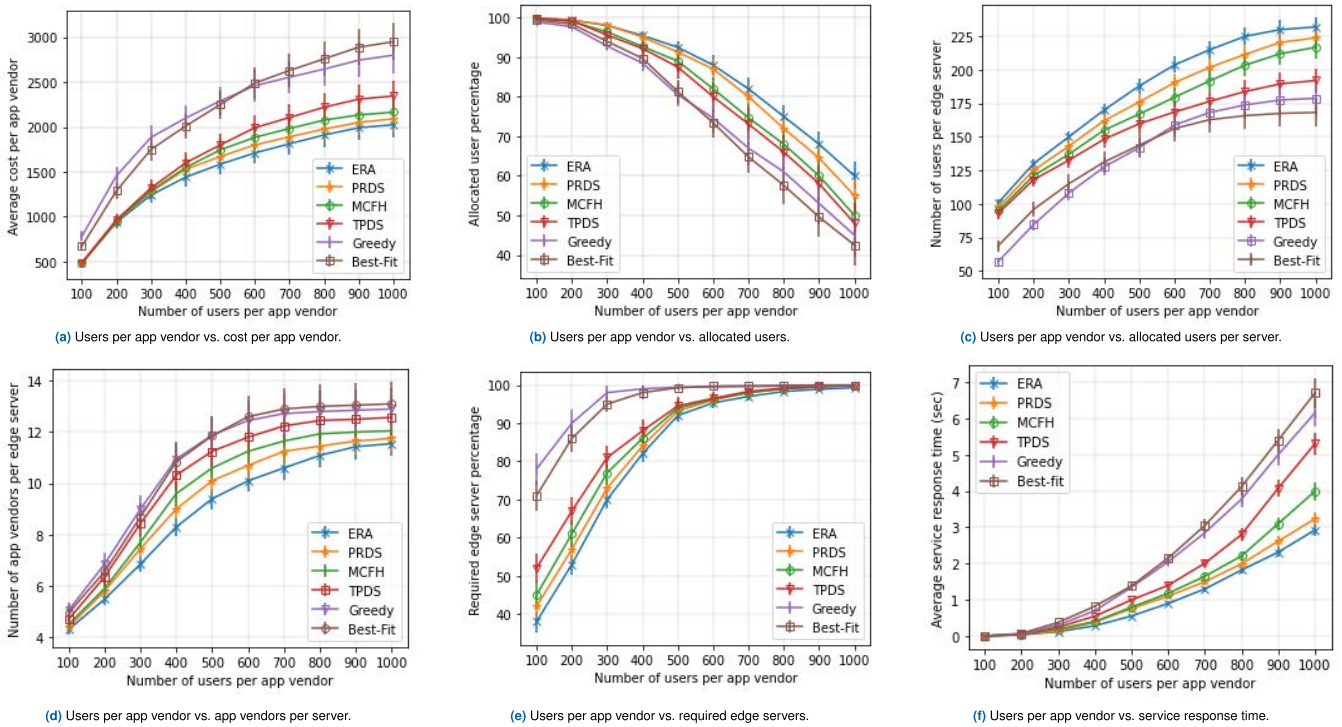
We evaluate the performance of the ERA algorithm by conducting experiments on three types of experimental settings given in Table 2. In each experiment, the ERA algorithm performance is compared with the baseline and state-of-the-art approaches in terms of the following metrics.

- *Cost per app vendor:* It denotes the average cost incurred by each app vendor for resource allocation on the edge servers.
- *Allocated users:* This metric represents the overall users of each app assigned to the edge servers.
- *App users per edge server:* This metric refers to the average number of users of each app assigned to every edge server. This calculation includes edge servers that serve at least one app user.
- *App vendors per edge server:* This represents the average number of apps deployed on each edge server.
- *Required edge servers:* This measurement refers to the edge servers used out of the total available edge servers.
- *Response time:* This is the average time interval between task submission and the first response received by app users. In our experiments, response time is calculated as the amount of waiting time in the queue for first-time CPU allocation and the time required to cache all of its computational task's pages.

Experiments 1, 2, and 3 correspond to set-1, set-2, and set-3 given in Table 2. Experiment 1, 2, and 3 evaluates the algorithm's performance with the different numbers of app users per vendor ranging from 100 to 1000, the different number of edge servers ranging from 10 to 100, and the different number of app vendors ranging from 5 to 50, respectively.

1) EXPERIMENT 1

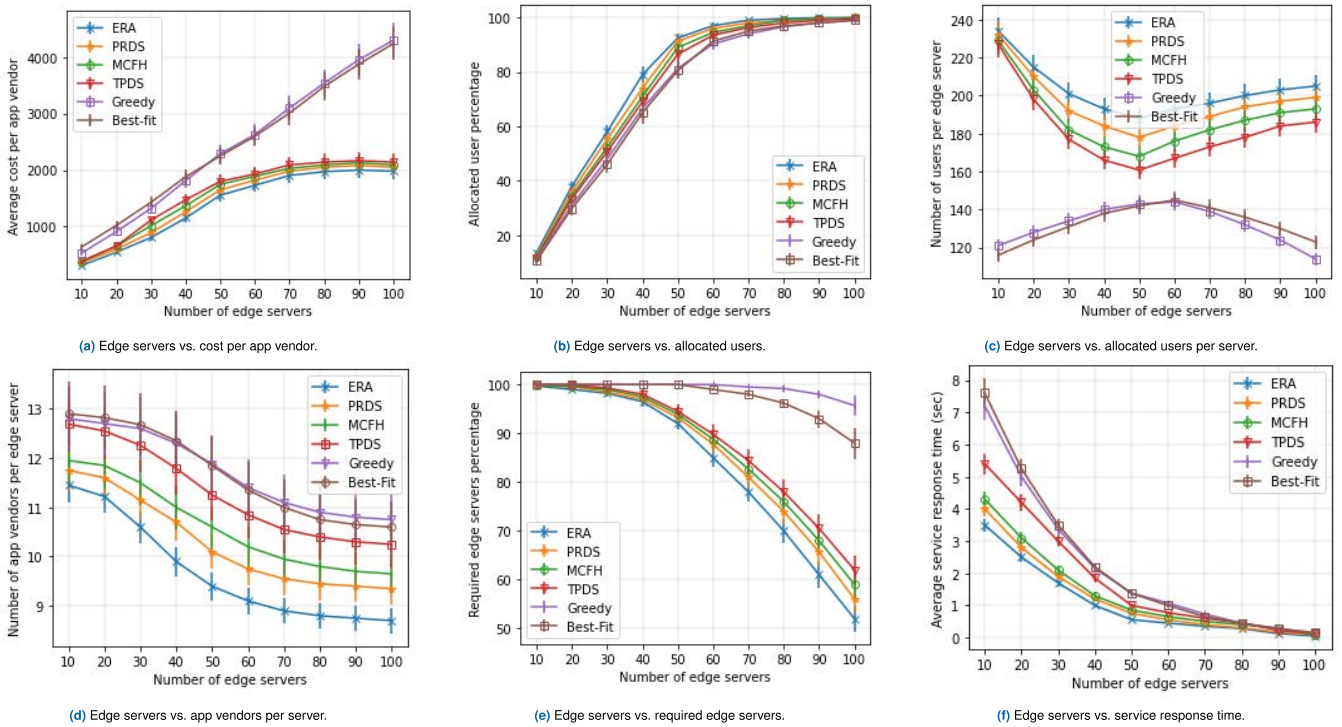
The experimental findings illustrated in Fig. 5 compare the algorithm's performance with different numbers of app users per vendor ranging from 100 to 1000, with the number of edge servers being constant at 50. The results depicted in Fig. 5a show that the average cost per app vendor increases when app users increase. ERA algorithm gives a more cost-effective solution than other alternatives as it appropriately allocates resources by utilizing the multi-tenancy. This happens because ERA maximizes the number of app users of similar services on each edge server that effectively utilizes the multi-tenant edge server. Fig. 5b shows that the overall allocated users to edge servers decreases with increasing the number of app users as proximity and capacity constraints of edge servers may leave app users unallocated. The ERA algorithm outperforms PRDS, TPDS, MCFH, Greedy, and Best-Fit. The reason is that ERA assigns more app users of a similar service to each edge server than to others, allowing for more app users to be served on a multi-tenant edge server with similar resources. As a result, various application components, such as data and programs, can be reused, leading to faster computation without increasing the computing resources. Allocated app users per edge server also follow a similar pattern as overall app user allocation. Therefore, each edge server serves more app users under the ERA algorithm compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 5c. The ERA algorithm enables



(a) Users per app vendor vs. cost per app vendor. (b) Users per app vendor vs. allocated users. (c) Users per app vendor vs. allocated users per server.

(d) Users per app vendor vs. app vendors per server. (e) Users per app vendor vs. required edge servers. (f) Users per app vendor vs. service response time.

FIGURE 5. Experimental results show the performance of ERA algorithm with varying number of app users per app vendor. In all the figures vertical line represents the error bar with a confidence interval of 95%.

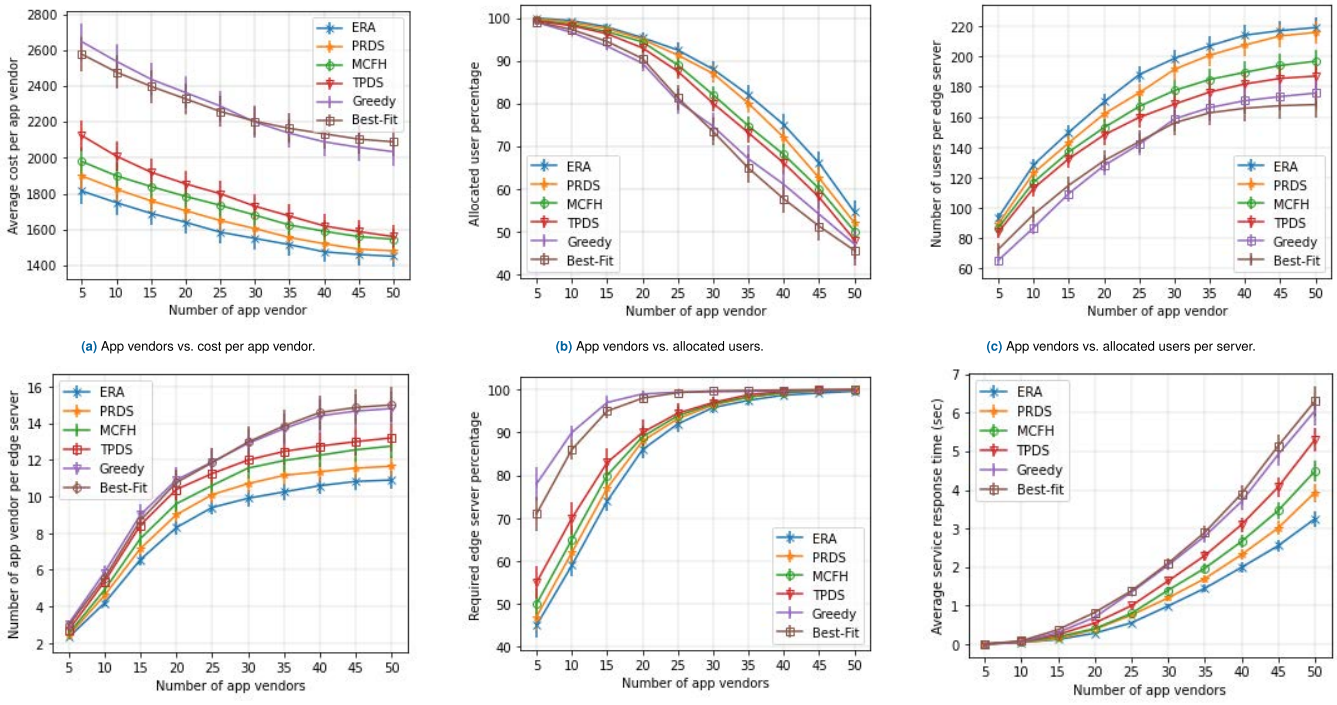


(a) Edge servers vs. cost per app vendor. (b) Edge servers vs. allocated users. (c) Edge servers vs. allocated users per server.

(d) Edge servers vs. app vendors per server. (e) Edge servers vs. required edge servers. (f) Edge servers vs. service response time.

FIGURE 6. Experimental results show the performance of ERA algorithm with varying number of edge servers. In all the figures vertical line represents the error bar with a confidence interval of 95%.

each edge server to serve more app users of similar services than other state-of-the-art and baseline techniques. Therefore, the number of app vendors per edge server is lower in the ERA algorithm than others, as shown in Fig. 5d. The experimental results in Fig. 5e illustrate that as the number of app users increases, so does the number of edge servers required.



(a) App vendors vs. cost per app vendor.
(b) App vendors vs. allocated users.
(c) App vendors vs. allocated users per server.
(d) App vendors vs. app vendors per server.
(e) App vendors vs. required edge servers.
(f) App vendors vs. service response time.

FIGURE 7. Experimental results show the performance of ERA algorithm with varying number of app vendors. In all the figures vertical line represents the error bar with a confidence interval of 95%.

The ERA performs well as it motivates the app vendors to switch on edge servers used by more services subject to proximity and capacity constraints. When the number of app users exceeds a threshold, all the approaches use the approximately maximum available edge servers at some point, e.g., 700 app users in our experiments. The results illustrated in Fig. 5f show that as the number of app users increases, the service response time to users increases. The ERA algorithm maximizes the number of app users per server who belong to similar services, enabling app users to utilize most of the cache contents and lowering the number of swapping for application programs and data. As a result, the resource overhead is reduced, and ERA outperforms other systems in our experiments.

2) EXPERIMENTS 2

In this experiment, the number of edge servers is varied from 10 to 100, and the app vendors and app users per vendor are kept constant at 25 and 500, respectively. Fig. 6a shows how the average cost of each app vendor changes as the number of edge servers increases. As only a few edge servers are available to app users at the start (10 edge servers), a smaller number of app users receive computational services. Consequently, the app vendors' cost is less. ERA algorithm outperforms PRDS, TPDS, MCFH, Greedy, and Best-Fit as the number of available edge servers increases. The reason is that the ERA algorithm reduces the required number of edge servers per app vendor by allocating resources appropriately to maximize the usage of multi-tenant edge servers. In each of the PRA, PRDS, MCFH, and TPDS, the cost incurred by each app vendor is approximately flattened when edge servers exceed a threshold, e.g., 70 in our experiment. This happens because the number of available edge servers exceeds the required edge servers, resulting in numerous edge servers being unutilized. On the other hand, Greedy and best -Fit does not allocate resources in order, resulting in a non-optimal number of edge servers in use. Therefore, the cost of each app vendor increases with increasing edge servers in Greedy and best-Fit.

The findings depicted in Fig. 6b show that the number of allocated app users increases as the availability of the edge servers increases in a given geographical area. When the number of edge servers is large enough, such as 80 in our experiment, all approaches almost allocate all app users to edge servers. ERA algorithm performs well as it allocates resources to more app users of a similar service on each edge server. Due to the same reason, the ERA algorithm allocates more app users per edge server compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 6c. These outcomes show that the resource utilization under the ERA algorithm is higher than the PRDS, TPDS, MCFH, Greedy, and Best-Fit. When edge servers are 10, the number of users per edge server in ERA, PRDS, MCFH, and TPDS is very high because the app vendors fully utilize a limited number of edge servers. In that case, many app users can not be allocated to any edge server. After that, the edge server utilization decreases to a certain point, e.g., 50 edge servers in our experiment, because resources on some edge

servers may be allocated to only a few app users. After a certain number of edge servers in a given geographical area, such as 50 in our investigation, the number of app users per edge server increases. The reason is that when edge servers exceed 50, the ERA, PRDS, MCFH, and TPDS reorder the resource allocation in a way that abandons the resources of edge servers used by very few app users and relocates app users among a limited number of edge servers. Hence, if the number of edge servers in a given geographical area exceeds a threshold, this limits the number of edge servers in use and results in high utilization of these edge servers by a fixed number of app users. The number of app users assigned to each edge server in the Greedy and Best-fit approach follows the opposite behavior compared to the ERA, PRDS, MCFH, and TPDS. This happens as Greedy and Best-fit do not allocate computing resources to app vendors in any order.

The results illustrated in Fig.6d present that the number of services per edge server decreases with the increasing number of edge servers. The reason is that the app vendors can assign more app users on single edge servers as the number of choices increases. Fig.6d shows that the ERA algorithm makes proper use of the multi-tenant edge servers. The findings in Fig. 6e illustrate that, up to a point, the number of edge servers used in each technique is nearly identical, e.g., 30 in our experiments, as there is a lower availability of edge servers. As described above, the overall required edge servers are lower in the ERA algorithm. The observations in Fig. 6f depict that service response time decreases as the availability of edge servers increases. The ERA algorithm works well because it maximizes cache content utilization while increasing data and software reuse.

3) EXPERIMENT 3

Fig. 7 shows the experimental results, which compare the algorithm's performance with a variation of app vendors ranging from 5 to 50, with app users per vendor and edge servers set at 500 and 50, respectively. The results depicted in Fig. 7a show that the average cost per app vendor decreases with an increase in app vendors as the fixed cost of the edge servers distributes among them. The solution achieved by the ERA algorithm is more cost-effective than other approaches as it allocates resources effectively to maximize the multi-tenancy use. The reason is that the ERA algorithm maximizes the number of app users of similar services on each edge server that effectively utilizes the multi-tenant edge server. Fig. 7b shows that the overall allocated users to edge servers decrease with app vendors. It happens because an increase in app vendors leads to an increase in app users, which causes edge servers to become unavailable owing to proximity and capacity restrictions. The ERA algorithm performs better than PRDS, TPDS, MCFH, Greedy, and Best-Fit. The reason is that ERA assigns more app users of a similar service to each edge server than to others, allowing for more app users to be served on a multi-tenant edge server with similar resources because of various application components, such as program and data, can be reused. Due to the same reason, in the ERA algorithm, each edge server can serve more app users compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 7c. Compared to other state-of-the-art and baseline methodologies, the ERA algorithm allows each edge server to serve more app users of similar services. Thus, the number of app providers per edge server is lower under the ERA algorithm, as shown in Fig. 7d. The findings in Fig. 7e show that when the number of app vendors increases, the number of edge servers required increases as well. The ERA performs well as it motivates the app vendors to switch on edge servers used by more services subject to proximity and capacity constraints. The results in 7f show that as the number of app vendors grows, so does the service response time to app users. The ERA algorithm performs well, as discussed above in Experiments 1 and 2.

VIII. CONCLUSION

In this paper, we addressed the problem of allocating multi-tenant edge computing resources to different IoT apps (services) vendors. We proposed an ERAGame, a game-theoretic approach that formulates the Edge Resource Allocation (ERA) problem as a potential game. The goal of the proposed game is to maximize resource utilization and increase app users assigned to Edge servers while minimizing costs incurred by app vendors. To accomplish this objective, the ERAGame employs the ERA algorithm for convergence to the Pure Nash Equilibrium (PNE) solution. This way, the ERA problem can be solved in a distributed manner. The ERA algorithm takes at most $\max_{q_{i,x}}(T \times q_{i,x})$ iterations to reach the PNE. We got bound $O(log\ n)$ for the price of stability of ERAGame under the ERA algorithm. We conducted extensive experiments and compared the ERA algorithm's performance with baseline and state-of-the-art approaches. The experimental findings validate that the ERA algorithm allocates the optimal bundle of computing resources by maximizing resource utilization and assigns the most app users to edge servers at a lower cost.

This study opened many new avenues for future investigation of the ERA problem. 1) the effects of mobility and trajectories of app users on ERAGame can be investigated. 2) Future research could look into app users' mobile participation in ERAGame, including existing app user departures and new app user arrivals. 3) ERAGame can be improved to accommodate the diverse capabilities needs of app users over time. 4) The impact on the quality of the user experience in ERAGame can be investigated.

REFERENCES

[1] G. Li, J. He, S. Peng, W. Jia, C. Wang, J. Niu, and S. Yu, "Energy efficient data collection in large-scale Internet of Things via computation offloading," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4176–4187, Jun. 2019.

[2] S. Kumar, A. Goswami, R. Gupta, S. P. Singh, and A. Lay-Ekuakille, "A game-theoretic approach for cost-effective multicast routing in the Internet of Things," *IEEE Internet Things J.*, early access, Apr. 1, 2022, doi: 10.1109/JIOT.2022.3164028.

[3] A. Boukerche, S. Guan, and R. E. De Grande, "A task-centric mobile cloud-based system to enable energy-aware efficient offloading," *IEEE Trans. Sustain. Comput.*, vol. 3, no. 4, pp. 248–261, Oct. 2018.

[4] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.
[5] J. Cohen, "Embedded speech recognition applications in mobile phones: Status, trends, and challenges," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2008, pp. 5352–5355.
[6] 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 *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2012, pp. 59–66.
[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[8] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.
[9] X. Zhang, Z. Li, C. Lai, and J. Zhang, "Joint edge server placement and service placement in mobile edge computing," *IEEE Internet Things J.*, early access, Nov. 13, 2021, doi: 10.1109/JIOT.2021.3125957.
[10] X. Chen, Y. Cai, L. Li, M. Zhao, B. Champagne, and L. Hanzo, "Energy-efficient resource allocation for latency-sensitive mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2246–2262, Feb. 2020.
[11] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "ULOOF: A user level online offloading framework for mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 17, no. 11, pp. 2660–2674, Nov. 2018.
[12] 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 Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.
[13] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.
[14] S. Batewela, C.-F. Liu, M. Bennis, H. A. Suraweera, and C. S. Hong, "Risk-sensitive task fetching and offloading for vehicular edge computing," *IEEE Commun. Lett.*, vol. 24, no. 3, pp. 617–621, Mar. 2020.
[15] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multi-user industrial Internet of Things through edge computing," *IEEE Internet Things J.*, early access, Oct. 27, 2021, doi: 10.1109/JIOT.2021.3123406.
[16] H. Hu, Q. Wang, R. Q. Hu, and H. Zhu, "Mobility-aware offloading and resource allocation in a mec-enabled iot network with energy harvesting," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17541–17556, Dec. 2021.
[17] Y. Zhang, X. Zhao, Z. Zhou, P. Qin, S. Geng, C. Xu, Y. Wang, and L. Yang, "Robust resource allocation for lightweight secure transmission in multicarrier NOMA-assisted full duplex IoT networks," *IEEE Internet Things J.*, vol. 9, no. 9, pp. 6443–6457, May 2022.
[18] J. Huang, M. Wang, Y. Wu, Y. Chen, and X. Shen, "Distributed offloading in overlapping areas of mobile edge computing for Internet of Things," *IEEE Internet Things J.*, early access, Jan. 20, 2022, doi: 10.1109/JIOT.2022.3143539.
[19] D. T. Nguyen, L. B. Le, and V. Bhargava, "Price-based resource allocation for edge computing: A market equilibrium approach," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 302–317, Jan./Mar. 2021.
[20] 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 Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Sep. 2020.
[21] P. Lai, Q. He, J. Grundy, F. Chen, M. Abdelrazek, J. G. Hosking, and Y. Yang, "Cost-effective app user allocation in an edge computing environment," *IEEE Trans. Cloud Comput.*, early access, Jun. 11, 2020, doi: 10.1109/TCC.2020.3001570.
[22] B. Jia, H. Hu, Y. Zeng, T. Xu, and Y. Yang, "Double-matching resource allocation strategy in fog computing networks based on cost efficiency," *J. Commun. Netw.*, vol. 20, no. 3, pp. 237–246, Jun. 2018.
[23] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1529–1541, Jul. 2021.
[24] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1277–1287, Jul. 2021.
[25] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *J. Parallel Distrib. Comput.*, vol. 132, pp. 190–203, Oct. 2019.
[26] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020, doi: 10.1109/ACCESS.2020.2981434.
[27] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. 8, pp. 10466–10477, 2020, doi: 10.1109/ACCESS.2020.2965620.
[28] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26652–26664, 2019, doi: 10.1109/ACCESS.2019.2900530.
[29] Y. Zhang, X. Dong, and Y. Zhao, "Decentralized computation offloading over wireless-powered mobile-edge computing networks," in *Proc. IEEE Int. Conf. Artif. Intell. Inf. Syst. (ICAIIS)*, Mar. 2020, pp. 137–140, doi: 10.1109/ICAIIS49377.2020.9194840.
[30] M. Aazam, M. St-Hilaire, C.-H. Lung, and I. Lambadaris, "MeFoRE: QoE based resource estimation at fog to enhance QoS in IoT," in *Proc. 23rd Int. Conf. Telecommun. (ICT)*, May 2016, pp. 1–5.
[31] 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 *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2018, pp. 230–245.
[32] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Oct./Dec. 2017.
[33] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.
[34] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 16, p. e3975, Aug. 2017.
[35] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1281–1290.
[36] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 207–215.
[37] 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 *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2019, pp. 86–101.
[38] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, S. Pang, H. Liu, Y. Qin, and P. Chen, "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2019, pp. 91–98.
[39] S. Ghosh and D. P. Agrawal, "A high performance hierarchical caching framework for mobile edge computing environments," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2021, pp. 1–6, doi: 10.1109/WCNC49053.2021.9417323.
[40] M. Sarra, B. Samia, S. Khaled, and D. Mehammed, "New caching system under uncertainty for mobile edge computing," in *Proc. 4th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Jun. 2019, pp. 129–134, doi: 10.1109/FMEC.2019.8795356.
[41] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5506–5519, Aug. 2018.
[42] R. Buyya and S. N. Srirama, *Fog and Edge Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2019.
[43] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. Workshop Power Aware Comput. Syst. OSDI*, 2008, pp. 1–5.
[44] G. Velkoski, M. Simjanoska, S. Ristov, and M. Gusev, "CPU utilization in a multitenant cloud," in *Proc. Eurocon*, Jul. 2013, pp. 242–249.
[45] D. Bozdag, U. Catalyurek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, "A parallel distance-2 graph coloring algorithm for distributed memory computers," in *Proc. Int. Conf. High Perform. Comput. Commun.* Berlin, Germany: Springer, 2005, pp. 796–806.

SUMIT KUMAR received the M.Tech. degree in computer science and engineering from the National Institute of Technology (NIT) Hamirpur, India, in 2011. He is currently pursuing the Ph.D. degree in edge computing enabled Internet of Things with applications in game theory with the Indian Institute Technology (BHU) Varanasi, India.

RUCHIR GUPTA (Senior Member, IEEE) received the Ph.D. degree in peer-to-peer networks from the Indian Institute of Technology Kanpur, India, in 2013. From 2017 to 2020, he was an Associate Professor with the Indian Institute Technology (BHU), Varanasi, India. He is currently working as a Professor with the School of Engineering, Jawaharlal Nehru University (JNU), Delhi. His research interests include peer-to-peer networks, social networks, game theory, NLP, and machine learning.

K. LAKSHMANAN received the Ph.D. degree in computer science and automation from the Indian Institute of Science (IISc), Bengaluru, India, in 2013. He has experience working as a Postdoctoral Fellow at the Indian Institute Technology Bombay, India, and the National University of Singapore, Singapore. He is currently an Assistant Professor of computer science and engineering at the Indian Institute of Technology (BHU) at Varanasi, Varanasi, India. His research interests include machine learning, stochastic optimization, and reinforcement learning.

VIPIN MAURYA received the B.Tech. degree in computer science engineering from the B.B.S. College of Engineering and Technology, Prayagraj, India, in 2018. He is currently pursuing the Ph.D. degree in wireless sensor network with the Indian Institute Technology (BHU) at Varanasi, Varanasi, India.