# Chapter 3

# App User Allocation

This chapter investigates a critical problem from the perspectives of the app vendor (IoT application provider) and its app users (IoT device users). It proposed a game-theoretic approach that allocates the app users to the edge servers while establishing a trade-off between the cost and Quality of Service (QoS).

## 3.1 Introduction

As illustrated in Fig 3.1, an edge server typically serves a small geographical area. An app user in the overlapped area can connect to any edge server that provides the app user-related service. Therefore, there are many ways to allocate app users to edge servers. If the number of app users allocated to an edge server exceeds the capacity of the edge servers, the QoS to app users may be compromised. On the other hand, if the number of app users assigned to an edge server is less than the edge server capacity, the edge server may be underutilized. Therefore, an inappropriate app user allocation may lead to the under-utilization of resources or degrade the QoS.

The app vendor pays the edge infrastructure providers for the hired resources. Their goal is to provide services to as many app users as possible while using the fewest number of edge servers possible. This results in lower costs and higher utilization of

hired resources on edge servers [42]. Multi-tenancy can reduce overall system costs by allowing more app users to offload their tasks to multi-tenant edge servers rather than a single-tenant edge server, reducing the number of edge servers in use [41, 117]. However, if the number of users assigned to a multi-tenant edge server exceeds a certain threshold, the QoS provided to app users can deteriorate as resource sharing overhead increases [41]. Therefore, app users must be efficiently allocated to multi-tenant servers to maximize resource utilization while maintaining QoS. This problem is referred to as the App User Allocation (AUA) problem.

The AUA problem is formulated as an app User Allocation Game (UAGame), which aims to provide the required QoS via load balancing on the edge servers while lowering system costs by maximizing the utilization of the edge server resources. The key is to fully utilize the multi-tenant architecture to serve the most significant number of app users with the fewest number of edge servers while maintaining the required QoS. In other words, our proposed approach establishes a balance between overall system cost and the app users' QoS. The UAGame also alleviates the burden of centralized optimization because it solves the problem in a decentralized way and employs an algorithm that converges faster to the solution. Our research makes the following significant contributions:

- The AUA problem is modeled as an optimization problem with the objective of minimizing overall cost while maintaining app users' QoS.
- A UAGame is introduced to solve the problem in a distributed manner, with app users simulated as players. The objective of this game is to find a stable optimum as a Pure Nash Equilibrium (PNE).
- A distributed AUA algorithm is proposed to achieve a PNE that establishes a trade-off between cost and QoS.
- To reduce the time complexity of finding the PNE using the AUA algorithm, the network topology of the edge servers is partitioned into a few groups.

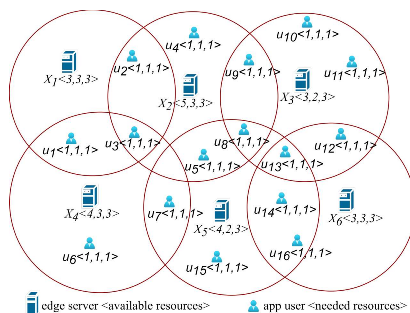



**Figure 3.1**: The example of IoT app user allocation to edge servers.

- The AUA algorithm convergence process is investigated by demonstrating that UAGame is a potential game.
- The optimality of PNE achieved by the AUA algorithm is theoretically investigated.
- An extensive simulation is performed to numerically analyze and compare the performance of the AUA algorithm with four state-of-the-art approaches.

The remaining chapter is organized as follows. Section 3.2 provides an illustration to motivate this study. Section 3.3 models the app user allocation as an optimization problem. Section 3.4 introduces the UAGame. Section 3.5 presents an algorithm to converge at PNE. Convergence and optimality analysis are discussed in Sections 3.6 and 3.7. Section 3.8 numerically evaluates the AUA algorithm.

## 3.2 Motivating Example

As a motivating example, this section will use a resource-hungry and latency-sensitive IoT application, such as a security management system. It can benefit from edge computing, which provides high computing capacity with low latency. In this application, multiple cameras monitor a specific area or building and generate megabytes of data per second. This system continuously offloads this data for processing and requires a quick response to deal with any adverse event. Fig. 3.1 shows six multi-tenant edge servers $\{X_1, \cdots, X_6\}$ with limited computing resources $< CPU, memory, bandwidth >$

available for security management app vendor. The app vendor allocates its app users $\{u_1, \cdots, u_{15}\}$ to these edge servers. The circle represents each edge server's geographical coverage area, and the edge server can serve app users in that area. For example, user $u_6$ can only be served by $X_4$. A capacity constraint must also be met to allocate an app user on the edge server. That means the available resources on the edge server are sufficient to meet the app user's requirements.

Minimizing the number of edge servers in use is the most cost-effective way of allocating app users to edge servers. The one possible app user allocation is ($X_2[u_2, u_4, u_5, u_8, u_9]$, $X_3[u_{10}, u_{11}, u_{12}]$, $X_4[u_1, u_3, u_7, u_6]$, $X_5[u_{13}, u_{14}, u_{15}, u_{16}]$) which reduces the number of utilized servers to four. In this allocation, the app users share the edge servers to a greater extent. For example, $X_5$ serves five app users. Multi-tenancy enables this allocation to increase the resource utilization of edge servers, allowing app users to share the computing resources. However, suppose the number of app users assigned to any edge server exceeds a certain threshold. In that case, the resource sharing overhead can be very high, lowering the QoS provided to users. For example, the waiting time for app users' tasks to get CPU for computation may increase, resulting in latency. On the other hand, another app user allocation to the edge servers may be ($X_1[u_2, u_3]$, $X_2[u_4, u_8, u_9]$, $X_3[u_{10}, u_{11}, u_{12}]$, $X_4[u_1, u_7, u_6]$, $X_5[u_5, u_{15}]$, $X_6[u_{13}, u_{14}, u_{16}]$). In this allocation, some app users are reallocated to $X_1$ and $X_6$, reducing the load on the overloaded edge servers. This allocation provides better QoS but increases the overall cost of the system because six edge servers are in use.

Although, there are many solutions to the AUA problem. However, it is not trivial to find a solution that forms a balance between the overall system cost and the QoS [2]. Furthermore, in a multi-tenancy environment, the AUA problem becomes more complicated because it affects the utilization of the edge servers, ultimately affecting the system cost. In real-world IoT applications, the scale of AUA problem can be much larger than in the example presented. Therefore, it is challenging to find an optimum

solution to allocate IoT app users to edge servers. Hence, a solution is needed for allocating app users effectively and efficiently across edge servers.

Our study uses a game-theoretic approach to allocate IoT app users to edge servers, which improves QoS by balancing the load on the edge servers and performing cost optimization. It solves the problem in a distributed manner. Furthermore, it gives each user the ability to make decisions in order to pursue their interests. In contrast to other game-theoretic approaches, our proposed approach finds the PNE in less time by partitioning the edge server set into groups and running the algorithm in parallel.

## 3.3 System Model

We consider an edge computing enabled IoT environment where $n$ app users $U = \{1, 2, 3, \cdots, n\}$ are using the services of an app vendor through $m$ edge servers $S = \{1, 2, 3, \cdots, m\}$. Each edge server $x$ has a vector $W_x = (W_x^d)$ representing its initial computing resources, where each dimension $d$, $d \in D = \{$ bandwidth, memory, storage, cpu $\cdots\}$, represents a resource type. Similarly, vector $\omega_i = (\omega_i^d)$ specifies the resources required by an app user $i$, where $d \in D$ such that $\omega_i^d$ denotes the resource requirement of type $d$.

*Proximity Constraint*: An app user $i$ can be assigned to an edge server $x$ only if $i$ falls within the $x$'s coverage range, denoted by $cov(x)$, as follows:

$$i \in cov(x), \ \forall i \in U, \ \forall x \in S. \tag{3.1}$$

Such an edge server $x$ is known as a neighbor edge server of app user $i$. The proximity edge server set for app user $i$, denoted by $A_i$ ($A_i \subseteq S$), contains all neighboring edge servers of $i$.

**Definition 3.1 (Allocation Decision)** *Given a neighboring edge server set $A_i$ ($A_i \subseteq S$) of an app user $i$, the allocation decision specifies the edge server allocated to the app*

*user $i$. It is represented by $a_i$ such that $a_i \in A_i$.*

**Definition 3.2 (Allocation Profile)** *Given an edge server set $S = \{1, \cdots, m\}$ and an app user set $U = \{1, \cdots, n\}$, a tuple that includes the allocation decisions (allocated edge servers) of all app users specifies an allocation profile. It is denoted by $a = (a_1, a_2, \cdots a_n)$, where each $a_i \in A_i$ (neighbor edge server set) and represents an app user $i$'s allocation decision.*

The cartesian product of all app users' neighboring edge server sets represents all possible allocation profiles such as $A = A_1 \times A_2 \times \cdots \times A_n$. The allocation profile in respect of a particular app user, $i$, can be written as $(a_i, a_{-i})$, where $a_{-i} = (a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$ represents the app users' allocation decisions except $i$. In this chapter, the terms $a$ and $(a_i, a_{-i})$ are used interchangeably.

*Capacity Constraint*: Given an app user set $U = \{1, \cdots, n\}$ and an edge server $x$, the capacity constraint for each resource type $d$ of edge server $x$ is such that:

$$\sum_{i \in R_x^a | R_x^a \subset U} \omega_i^d \leq W_x^d, \ \forall d \in D, \forall x \in S, \tag{3.2}$$

here set $R_x^a$ contains app users allocated to a server $x$ at allocation profile $a$, where each $\omega_i^d \geq 0$, $\forall d \in D$ and $\forall i \in U$. An app user $i$ can only be allocated to an edge server $x$ if $x$'s available resources meet $i$'s resource requirements.

We assume the location of app users remains fixed during the execution of the server selection process as in [41, 42].

### 3.3.1 Network Topology of Edge Servers

Edge servers, in general, are static entities that provide services to app users within the coverage areas indicated by the circles in Fig. 3.2a. Edge server deployment is represented by a graph $G(S, E)$, where vertex set $S$ contains the edge servers and set $E$ contains edges among them. An edge exists between two edge servers if their coverage

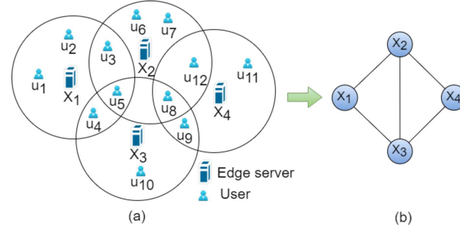



**Figure 3.2**: The example of (a) the deployment of edge servers and (b) their network topology.

areas overlap, as shown in Fig. 3.2b. This graph depicts the edge server network topology provided by edge infrastructure providers to app vendors, enabling them to move their app-related services from one server to another as needed [118].

### 3.3.2 Resource Utilization Model

Multi-tenant server architectures allow for greater resource utilization on cloud and edge servers than single-tenancy [41, 119]. In a multi-tenancy, a single instance of a software application running on an edge server handles various app users simultaneously, where each app user is a tenant. Multi-tenancy offers shared tenancy across different service providers like Google Cloud, Amazon Web Services, Microsoft Azure, etc. In [120], the result of the experiments show the approximated CPU utilization of a multi-tenant server $x$ as follows:

$$Z_{cpu}(x) = -log_y(|R_x^a|), \tag{3.3}$$

where $R_x^a$ ($|R_x^a| > 1$) is an app user set allocated to server $x$ at allocation profile $a$, here $|R_x^a|$ represent the cardinality of $R_x^a$, and parameter $y$ controls the growth of CPU utilization based on the average computational task size offloaded by users. As in studies [41, 120], $y$ ($0 < y < 1$) is calculated by the average computational task size offloaded on the edge server and grows in proportion to the average task size. It implies that if the average computational size is large, CPU utilization on the edge server increases rapidly with the increasing number of app users, and vice versa for a smaller average task size.

The CPU usage of the server usually increases with increasing app users in $R_x^a$ because multi-tenancy allows for resource sharing on the server to a higher degree. Up to some point, the overall CPU usage of a multi-tenant server is outperformed by the usage of multiple single-tenant servers combined [119]. But after a threshold, CPU utilization decreases rapidly due to the higher overhead of resource sharing. As a result, the QoS gets degraded for the app users. The storage and memory usage also follow the same pattern as CPU utilization [41, 119]. We assume that other computing capacity shared in a multi-tenant environment also follows a similar pattern as CPU, memory, and storage. Thus, the resources usage of a multi-tenancy server is calculated as:

$$Z_d(x) = -log_{y_d}(|R_x^a|), \tag{3.4}$$

where $d \in D$, and the average computational task size calculates the value of $y_d$. The size of the computational task has a different impact on each resource type $d$. Therefore, $y_d$ is distinct for each resource type $d$.

If the computing capacity of edge server $x$ for resource type $d$ is $W_x^d$ units, the average utilization of each unit can be calculated as follows:

$$Z_d(x) = -log_{y_d}(\frac{|R_x^a| \cdot \delta_d}{W_x^d}), \tag{3.5}$$

where $\delta_d$ is number of average units of resource type $d$ required by each app user In the worst-case scenario, the computational tasks offloaded by app users of a similar app will be of maximum size.

### 3.3.3 Edge Server Cost Model

The computing capacities of edge servers are rented out using a pricing model by the edge infrastructure providers. Several infrastructure providers, including Salesforce, Azure, and AWS, use the pay-as-you-go pricing model, which is investigated in this

chapter. This pricing model defines the cost based on the resource utilization calculated by Eq. 3.5. Thus, the app vendor's cost for hiring $W_x = (W_x^d)$ resources from an edge server $x$ can be computed as follows:

$$\zeta_x(R_x^a) = \sum_{d \in D} h_d \cdot Z_d(x).W_x^d, \tag{3.6}$$

where $h_d$ is the weight given to resource type $d$, which determines $d$'s priority, and $\sum_{d \in D} h_d = 1$. The edge server cost is the sum of the cost of all resources. The edge server cost incurred by the app vendor is the sum of the cost of all resources. Like $Z_d(x)$, function $\zeta_x(R_x^a)$ is also a non-decreasing concave function. The number of app users allocated to server $x$, as well as the size of their computational tasks, affect this cost function. The size of the task offloaded by each app user varies, so their contribution to the cost of edge servers is unequal. The rate of increase in the cost incurred by the app vendor decreases as the number of app users on the edge server increases because the cost function of an edge server is concave. As a result, this cost model encourages multi-tenancy.

**Marginal cost** Suppose app users in $R_x^a$ are assigned to edge server $x$; consequently, $x$'s cost will be $\zeta_x(R_x^a)$ from Eq. 3.6. At this moment, if an app user $i$ is assigned to $x$, the marginal contribution to the cost of $x$ due to the allocation of $i$ is,

$$\zeta_x(R_x^a \cup \{i\}) - \zeta_x(R_x^a), \tag{3.7}$$

where $\zeta_x(R_x^a \cup \{i\})$ is the cost of $x$ incurred by the app vendor after the allocation of $i$ to $x$.

### 3.3.4 App Users Usage Cost Model

Cost $\zeta_x(R_x^a)$ for an edge server $x$ is distributed among the app users (in $R_x^a$) who offload their computational tasks to $x$. This cost is referred to as the app users' usage cost,

and $\varrho_i(a)$ denotes it for each app user $i$. The Shapley value solution concept, which fairly distributes the cost of an edge server among app users, can be used [52]. The fair division of costs implies that the Shapley value meets the following conditions: 1) The entire cost of the edge server is distributed among the app users —nothing is wasted. 2) App users who offload similar tasks (identical app users) to the edge server incur equal portions of the edge server cost. 3) An app user who does not offload the task to the edge server incurs no cost.

The usage cost of an app user $i \in R_x^a$ at allocation profile $a$ can be calculated using the Shapley value as follows:

$$\varrho_i(a) = \sum_{R \subseteq R_x^a \setminus \{i\}} \frac{|R|!(|R_x^a| - |R| - 1)!}{|R_x^a|!} (\zeta_x(R \cup \{i\}) - \zeta_x(R)), \tag{3.8}$$

where $R$ is a subset of $R_x^a$ not containing app user $i$, and $(\zeta_x(R \cup \{i\}) - \zeta_x(R))$ is the marginal contribution by app user $i$ to the cost of server $x$, and the sum of marginal contribution $(\zeta_x(R \cup \{i\}) - \zeta_x(R))$ extends over all subsets $R$ of $R_x^a \setminus i$. The Shapley value allocates the average expected marginal cost to each app user by considering all possible combinations.

### 3.3.5 App Users QoS-degradation Cost Model

Quality of service (QoS) is a description of a service's overall performance that can be quantified in several parameters such as reliability, availability, software response time, latency, etc. Each application has different preferences for the QoS parameters, e.g., an emergency service application requires a minimum delay and lower packet loss. The number of app users allocated to any edge server affects the various QoS parameters, such as queuing time, resource sharing overhead, traffic on channel, etc. Therefore, when the number of app users on an edge server exceeds a certain threshold, QoS rapidly degrades. App users perceive the decline in QoS as a cost to themselves, denoted by

$q_x(R_x^a)$, as follows.

$$q_x(R_x^a) = \sum_{\lambda \in \Pi} I_\lambda \cdot G_\lambda(R_x^a), \tag{3.9}$$

where each $\lambda \in \Pi = \{$ availability, software response time, latency $\cdots \}$ is a QoS parameter, $I_\lambda$ is the assigned weight that specifies the preference of $\lambda$, such as $\sum_{\lambda \in \Pi} I_\lambda = 1$, and $G_\lambda(\cdot)$ is a normalized function. $G_\lambda(\cdot)$ expresses different characteristics of different QoS parameters. The following normalized functions are used to model QoS decline: 1) the exponential form [19, 121], 2) the logarithmic form [121], 3) the sigmoid form [19], and 4) the linear piece-wise form [122]. This chapter employs an exponential form that rapidly increases after app users exceed a certain threshold. The following is the exponential form of function $G_\lambda(R_x^a)$:

$$G_\lambda(R_x^a) = e^{\left(\alpha_\lambda \cdot \sum_{d \in D} \frac{\theta_d^\lambda \cdot Z_d(x)}{W_x^d} - \gamma_\lambda\right)}, \tag{3.10}$$

where $0 < \alpha_\lambda < 1$ is a tunable parameter that controls the function's growth rate, $\gamma_\lambda > 0$ controls where the function's growth should begin, and $\theta_d^\lambda$ indicates how much resource type $d$ affects QoS parameter $\lambda$ such that $\sum_{d \in D} \theta_d^\lambda = 1$. For example, CPU, memory, and bandwidth each impact the delay parameter of QoS differently.

### 3.3.6 Optimization Model

At allocation profile $a$, each app user $i$ assigned to an edge server $x$ incurs the cost of task offloading, denoted by $c_i(a)$. This cost comprises two components: (i) usage cost and (ii) QoS-degradation cost. Thus,

$$c_i(a) = \varrho_i(a) + q_x(R_x^a). \tag{3.11}$$

Given edge servers $S = \{1, \cdots, m\}$, and app users $U = \{1, \cdots, n\}$, the optimization model for an AUA problem is formally expressed as follows:

$$\min_{a \in A} \sum_{i \in U} c_i(a), \tag{3.12}$$

subject to:

$$i \in cov(x), \ \forall i \in U, \forall x \in S, \tag{3.13}$$

$$\sum_{i \in R_x^a | R_x^a \subset U} \omega_i^d \leq W_x^d, \ \forall d \in D, \forall x \in S, \tag{3.14}$$

$$\omega_i^d \geq 0, \ \forall i \in U, \forall d \in D. \tag{3.15}$$

Objective (3.12) reduces the costs of app users while improving their QoS and establishes a trade-off between usage cost and QoS. This study aims to find an allocation profile $a \in A$ that achieves this objective. Constraints (3.13) ensure that a user $i \in U$ can only be assigned to a server $x \in S$ if $x$ covers $i$. Constraint (3.14) ensures that the computing resource requirements of app users assigned to an edge server $x$ do not exceed the available computing resources of that edge server. Constraint (3.15) states that no app user needs a negative computing resource amount.

## 3.4 User Allocation Game

In this section, we present a User Allocation Game (UAGame) to solve the AUA problem. The number of users allocated to a server affects all users' usage cost and QoS, as represented in Eqs. 3.8 and 3.9. As a result, assigning or removing an app user from a server impacts the cost and QoS of that app user and other app users on that server. Therefore, there is ***strategic interaction*** among app users, as each app user's outcome is dependent on their own and other app users' decisions. Hence, we can model this strategic interaction as an UAGame. This game aims to find an allocation profile (having one allocation decision for each user) that cost-effectively distributes app users among edge servers while maintaining QoS. The app users make their decisions by following the game's rules. Given an allocation profile $a_{-i}$ of other app users except for $i$, app user $i$'s objective in UAGame is to make a decision $a_i$ that minimizes its cost in response to other app users' decisions:

$$\min_{a_i \in A_i} c_i(a_i, a{-}i), \tag{3.16}$$

where set $A_i$ contains $i$'s neighboring servers that geographically cover $i$.

Based on (3.16), the AUA problem can be formulated as a UAGame denoted by the tuple $(U, \{A_i\}_{i \in U}, \{c_i(a)\}_{i \in U, a \in A_1 \times \cdots \times A_n})$, where set $U = \{1, 2, \cdots, n\}$ contains $n$ app users acting as players, set $A_i$ (neighbor edge server set) contains app user $i$'s finite allocation decisions, $c_i(a)$ refers to app user $i$'s cost at allocation profile $a$. In this game-theoretic approach, each app user makes the best allocation decision according to their interests in response to other app users' decisions. The system moves gradually toward allocation profile $a^* = (a_1^*, a_2^*, \cdots, a_n^*)$, a Pure Nash Equilibrium (PNE), at which the cost tuple $(c_1(a^*), c_2(a^*), \cdots, c_n(a^*))$ of all app users is stably optimum.

**Definition 3.3** *[Best Allocation Decision] Given the allocation profile $a_{-i}$ of app users except user $i$, a best allocation decision of app user $i$, denoted by $a_i'$, minimizes its cost, which is defined as:*

$$BAD_i(a_{-i}) = \{a_i' \in A_i : c_i(a_i', a_{-i}) \leq c_i(a_i, a_{-i}), \forall a_i \in A_i\}. \tag{3.17}$$

**Definition 3.4 (Pure Nash Equilibrium (PNE))** *In UAGame, an allocation profile $a^* = (a_i^*, a_{-i}^*)$ is a PNE if no app user can minimize its cost further by unilaterally changing its allocation decision, i.e.,*

$$c_i(a_i^*, a_{-i}^*) \leq c_i(a_i, a_{-i}^*), \quad \forall a_i \in A_i, \forall i \in U. \tag{3.18}$$

Finding out whether the system reaches at least one PNE is an important part of this research. Section 3.6 analyze the convergence to the PNE in UAGame.

## 3.5 Distributed User Allocation

This section presents a distributed App User Allocation Algorithm (AUA algorithm) for achieving PNE convergence. In UAGame, users make their allocation decisions in

response to other users. Users see these decisions as progress from the perspective of usage cost minimization and QoS improvement. If these decisions are made concurrently, the outcome of each decision may not be a gain because their decisions influence each other's costs. Hence, only a single user must make a decision at a time in response to other users' decisions to ensure improvement with every decision. However, reaching PNE could take considerable time.

For faster convergence to PNE, we divide the edge server network topology into a few groups and feed them to the proposed AUA algorithm. This grouping allows the algorithm to run in parallel within each group without affecting the quality and existence of PNE in UAGame, resulting in quick convergence. Each group has a property that decisions made on one server are unaffected by decisions made on other servers in that group. However, in UAGame, the PNE is irrespective of the number of groups and the grouping process.

### 3.5.1 Edge Server grouping (System Preprocessing)

#### 3.5.1.1 Desired Properties of groups

grouping aims to run the AUA algorithm on all edge servers in each group in parallel. The edge server grouping is performed based on the distance between edge servers as determined by the edge server network topology. In each group, the distance between any two edge servers must be greater than two (number of edges). For example (Fig. 3.3), the shortest path length between edge servers $X_2$ and $X_{11}$ is greater than two, so $X_2$ and $X_{11}$ can be grouped together. Because the distance between any two edge servers in a group exceeds 2, app users' decisions on one edge server have no direct impact on allocation decisions on other edge servers in that group. The following example from Fig. 3.3 demonstrates this. Suppose edge servers $X_2$ and $X_{11}$ are in the same group and their coverage areas overlap with the edge servers in $\{X_1, X_3, X_5, X_6\}$ and $\{X_8, X_{12}, X_{15}\}$, respectively; $\{X_1, X_3, X_5, X_6\} \cap \{X_8, X_{12}, X_{15}\} = \emptyset$. Any app

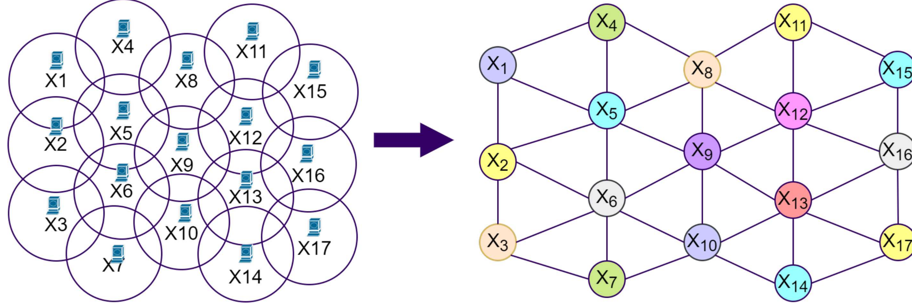



**Figure 3.3**: The example of edge server grouping.

user allocated to $X_2$ that falls in the overlap area can only be reallocated to an edge server in $\{X_1, X_3, X_5, X_6\}$. Similarly, app users allocated to $X_{11}$ can be reallocated to $\{X_8, X_{12}, X_{15}\}$. Suppose app users allocated to $X_2$ and $X_{11}$ are reallocated simultaneously. In that case, the reallocation decisions do not directly affect each other's reallocated app users as $\{X_1, X_3, X_5, X_6\} \cap \{X_8, X_{12}, X_{15}\} = \emptyset$.

As a result, every group has the property that user allocation decisions on a group server have no direct impact on allocation decisions made on other edge servers in the same group. Our proposed algorithm can be run in parallel on all edge servers in a group because of this property, which reduces the time complexity to converge at PNE. Hence, the edge server set $S$ is partitioned into groups as $T = \{T_1, T_2, \cdots, T_M\}$, where any $T_j$ denotes the $j$th group, and $M$ is the number of groups. Each group has the following characteristics:

- $distance(x_{k_1}, x_{k_2}) > 2$, for every pair $(x_{k_1}, x_{k_2}) \in T_j$, for every $T_j \in T$
- $T_j \cap T_l = \emptyset$, for every $T_j, T_l \in T$

#### 3.5.1.2 groups Formation

The edge server groups are formed with the help of the network topology graph $G(S, E)$ of edge servers. The *distance-2 graph coloring* problem is analogous to the edge server grouping problem. This coloring of graph G is a mapping from $S$ to $\{1, 2, \cdots, M\}$ in which every two edge servers, separated by at most two edges, get different integer numbers from $\{1, \cdots, M\}$, where every $j \in \{1, \cdots, M\}$ represents a distinct color.

The edge servers, which get the number $j \in \{1, \cdots, M\}$ are included in a group $T_j$. As shown in Fig. 3.3, edge servers $\{X_5, X_{14}, X_{15}\}$ represented in blue are part of the same group. This chapter employs a heuristic distance-2 graph coloring approach to create the desired groups [123]. With set $E$ of edges among the $m$ edge servers, this approach takes $O(m \times |E|)$ time complexity to create required group. 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.

### 3.5.2 App User Allocation Algorithm

In UAGame, each app user makes its own allocation decision to reduce its cost in response to other app users. Due to which all the app users constantly search for their following best allocation decision and gradually move towards a PNE. Given $S = \{1, \cdots, m\}$ and $U = \{1, \cdots, n\}$, the UAGame uses an iterative procedure described in Algorithm 3.1 to converge at the PNE. An arbitrary allocation profile $a = (a_1, a_2, a_3, \cdots a_n)$ is initialized to start this iterative process, and each edge server $x \in S$ creates its app user list $R_x^a$ at allocation profile $a$ (Lines 1-2). After initialization, an iterative process begins (Line 4-20). Algorithm iterations are denoted by $t(t = 1, 2, 3, \cdots)$, and $a(t)$ specifies the allocation profile at iteration $t$. The AUA algorithm uses the following messages in each iteration:

- $PARTICIPATE_x$: Any edge server $x$ send this message to the app users who are allocated to $x$. It notifies the app users to participate in the current iteration.
- $COST_REQ_i^x$: An app user $i$ requests the edge server $x$ to know how much cost $i$ will incur if $i$ will be allocated to $x$. This message contains the task size of app user $i$.
- $COST_RESPONSE_x^i$: The edge server $x$ sends a message to $i$ that contains the how much cost $i$ will incur if $i$ offloads the task on $x$. This cost is calculated from the Eq. 3.11.

- $ADD_REQ_i^x$: An app user $i$ sends a request to edge server $x$ to be allocated. After receiving this request, $x$ can add app user $i$ to its user list $R_x^a$.
- $ACK_x^i$: Edge server $x$ sends an acknowledgment to app user $i$, which notifies that $x$ has added $i$ to its user list.
- $REMOVE_REQ_x^i$: App user $i$ sends a request to edge server $x$. After receiving this request, $x$ removes app user $i$ from its user list $R_x^a$.

At the beginning of every iteration of AUA algorithm, a group $T_l \subseteq S$ is opted in a decentralized manner (Line 4) using message synchronization approach [124]. All the calculation for each app user on every edge server within group $T_l$ are done in parallel in every iteration (Line 5-28). Each edge server $x \in T_l$ sends $PARTICIPATE_x$ message (Line 5-6). After receiving the notification for participating, each participant app user $i$ sends a $COST_REQ_i^z$ to each edge server $z$ that is in its allocation decision set $A_i$ (Line 7-8). Each app user $i$ which previously sent the $COST_REQ_i^z$ messages to the edge servers $z$ will get the $COST_RESPONSE_z^i$ based on the app user's task size from the requested edge servers (Line 9-11). Based on the cost value received from edge servers, each participant app user $i$ finds its best allocation decision $a_i'$ according to Definition 3.3 that incurs the lowest cost in response to other app users' allocation decisions $a_{-i}(t)$ (Line 12). Lines 13-26 reallocate each participant $i$ from current edge server $a_{-i}(t)$ to new edge server $a_i'$. Every $i$ whose current allocation decision is not the best allocation decision sends an $ADD_REQ_i^{a_i'}$ to the corresponding edge server $a_i'$ (Line 13-14). If any edge server $a_i'$ receives requests from multiple app users, it accepts only one app user's request whose incurs the lowest cost, and adds that app user into its user list $R_{a_i'}^{a(t)}$ (Line 15-19). Each edge server $a_i'$ that accepts $ADD_REQ_i^{a_i'}$ sends an acknowledgment $ACK_{a_i'}^i$ (Line 20). Then, each app user $i$ that has received the acknowledgment sends a $REMOVE_REQ_i^{a_i(t)}$ request to its current edge server $a_i(t)$ (lines 23-24), after which $a_i(t)$ removes that app user from its user list (Line 25). In this way, iterations of AUA algorithm repeat for every group $T_l \subset S$. If UAGame is

**Algorithm 3.1:** App User Allocation Algorithm

**Input**: $U, \{A_1, A_2, \cdots, A_n\}, T$
**Output**: Pure Nash Equilibrium $a$

```
1  Set the allocation profile to random a ← (a_1, ··· , a_n)
2  each server x ∈ S do R_x^a ← {i | i is a user of x at a}
3  repeat
4  |  election of a group T_l for iteration t
5  |  for every x ∈ T_l do
6  |  |  edge server x transmit a PARTICIPATE_x notification to app users
   |  |  allocated to it
7  |  |  for every app user i allocated to x do
8  |  |  |  app user i sends COST_REQ_i^z to every edge server z ∈ A_i
9  |  |  |  for every edge server z ∈ A_i do
10 |  |  |  |  z sends COST_RESPONSE_z^i to i
11 |  |  |  end
12 |  |  |  i finds a'_i = BAD_{a_i(t)}(a_{-i}(t)) that incurs the lowest cost to i across A_i
   |  |  |  as in Definition 3.3
13 |  |  |  if a_i(t) ≠ a'_i then
14 |  |  |  |  i sends ADD_REQ_i^{a'_i} to edge server a'_i
15 |  |  |  |  if a'_i gets request from multiple app users then
16 |  |  |  |  |  if i's cost on a'_i is not the lowest among them then a'_i rejects
   |  |  |  |  |  the request
17 |  |  |  |  end
18 |  |  |  |  else
19 |  |  |  |  |  a'_i adds i into the app user list R_{a'_i}^{a(t)}
20 |  |  |  |  |  a'_i sends ACK_{a'_i}^i the app user i
21 |  |  |  |  end
22 |  |  |  end
23 |  |  |  if i gets ACK_{a'_i}^i then
24 |  |  |  |  i sends REMOVE_REQ_i^{a_i(t)} to a_i(t)
25 |  |  |  |  a_i(t) removes i from app user list R_{a_i(t)}^{a(t)}
26 |  |  |  end
27 |  |  end
28 |  end
29 until no more allocation decision updates needed
```

not in PNE, then again, iterations of AUA algorithm are repeated for every group, and the same process continues until PNE is achieved in UAGame. The solution will include the allocation decision of each app user at the PNE. In our algorithm, most of the computations take place at the edge servers.

## 3.6 Game Property

This section investigates the convergence to a PNE in UAGame using the AUA algorithm. To examine the existence of the PNE, we need to observe the AUA algorithm's convergence process, which can be accomplished by establishing the UAGame as a potential game. The following is the potential game:

**Definition 3.5 (Potential Game)** *If there is a potential function $\Phi(a)$ such that,*

$$c_i(a'_i, a_{-i}) - c_i(a_i, a_{-i}) \leq 0 \Rightarrow \Phi(a'_i, a_{-i}) - \Phi(a_i, a_{-i}) \leq 0,$$
$$\text{for every } i \in U, a_i, a'_i \in A_i \text{ and } a_{-i} \in \prod_{l \neq i} A_l, \tag{3.19}$$

*for a game, then this is a potential game.*

The overall potential of the UAGame can be determined by maintaining each edge server's potential. An app user who is assigned to a server incurs two costs: usage and QoS-degradation. These two costs affect the total potential of any edge server in different ways. Therefore, we calculate the potential of edge servers for both costs separately. Let $\Phi_{\zeta_x}$ and $\Phi_{q_x}$ denote the edge server $x$'s potential due to usage and QoS degradation costs, respectively. Thus, an edge server $x$'s total potential at an allocation profile $a$, denoted by $\Phi_x(a)$, is determined as:

$$\Phi_x(a) = \Phi_{\zeta_x} + \Phi_{q_x}. \tag{3.20}$$

**Calculation of $\Phi_{\zeta_x}$** We use the Shapley value concept to distribute the cost $\zeta_x(R^a_x)$ of a server $x$ among $R^a_x$ app users. For the Shapley value concept, there exists a unique potential function $\Phi_{\zeta_x}$ to calculate the potential of server $x$ due to the usage cost [125, 126], which is given as follows:

$$\Phi_{\zeta_x}(R^a_x, \zeta_x) = \frac{1}{|R^a_x|}\left[\zeta_x(R^a_x) + \sum_{i \in R^a_x} \Phi_{\zeta_x}(R^a_x \setminus \{i\}, \zeta_x)\right]. \tag{3.21}$$

It recursively determines the potential $\Phi_{\zeta_x}(R^a_x, \zeta_x)$ of an edge server, where $\Phi_{\zeta_x}(\emptyset, \zeta_x) = 0$. Function $\Phi_{\zeta_x}$ has some properties [126, 127] as:

$$\Phi_{\zeta_x}(R^a_x, \zeta_x) - \Phi_{\zeta_x}(R^a_x \setminus \{i\}, \zeta_x) = \varrho_i(a), \tag{3.22}$$

$$\sum_{i \in R^a_x} \Phi_{\zeta_x}(R^a_x, \zeta_x) - \Phi_{\zeta_x}(R^a_x \setminus \{i\}, \zeta_x) = \zeta_x(R^a_x), \tag{3.23}$$

where $\Phi_{\zeta_x}(R^a_x, \zeta_x) - \Phi_{\zeta_x}(R^a_x \setminus \{i\}, \zeta_x)$ is the marginal contribution of an app user $i$ that coincides with the $i$'s Shapley value $\varrho_i(a)$.

**Calculation of $\Phi_{q_x}$** Given an allocation profile $a$, the potential value $\Phi_{q_x}$ of a server $x$ as a result of QoS-degradation cost $q_x(R^a_x)$ is calculated as follows:

$$\Phi_{q_x}(R^a_x, q_x) = q_x(R^a_x) + \Phi_{q_x}(R^a_x \setminus \{i\}, q_x). \tag{3.24}$$

Starting with $\Phi_{q_x}(\emptyset, q_x) = 0$, Eq.3.24 calculates $\Phi_{q_x}(R^a_x, q_x)$ recursively.

**Overall potential $\Phi(a)$ of UAGame** Given an edge server set $S = \{1, \cdots, m\}$, the overall potential $\Phi(a)$ of UAGame at an allocation profile $a$ is determined as follows:

$$\Phi(a) = \sum_{x \in S} \Phi_x(a) = \sum_{x \in S} \Phi_{\zeta_x}(R^a_x, \zeta_x) + \Phi_{q_x}(R^a_x, q_x). \tag{3.25}$$

The overall potential of UAGame is the sum of the potentials of all edge servers.

**Theorem 3.1** $\Phi(a)$ *is a potential function and UAGame is potential game.*

**Proof:** If the potential function satisfies the definition given in Eq.3.19, the UAGame is a potential game. Under the AUA algorithm, a user $i$ updates his allocation decision (reallocates) from an edge server $a_i$ to $a'_i$ at an allocation profile $a = (a_i, a_{-i})$, only if $i$'s cost decreases. Suppose $R^a_{a_i}$ and $R^a_{a'_i}$ are users' list of edge servers $a_i$ and $a'_i$ before changing the allocation decision, respectively. The change in the cost of $i$ due to the

deviation from edge server $a_i$ to $a_i'$ is calculated from Eq. 3.11 as:

$$\begin{aligned}\Delta c_i &= c_i(a_i', a_{-i}) - c_i(a_i, a_{-i}) \\ &= \varrho_i(a_i', a_{-i}) + q_{a_i'}(R^a_{a_i'} \cup \{i\}) - \varrho_i(a_i, a_{-i}) - q_{a_i}(R^a_{a_i}) \\ &< 0 \end{aligned} \tag{3.26}$$

The potential function value is changed by,

$$\Delta\Phi = \text{increase in } \Phi - \text{decrease in } \Phi = \Phi_{inc} - \Phi_{dec} \tag{3.27}$$

When app user $i$ updates his allocation decision, $i$ is reallocated from server $a_i$ to $a_i'$. Due to the app user reallocation, edge server $a_i$'s potential decreases, and $a_i'$'s potential increases. Hence, edge servers $a_i$ and $a_i'$ contribute to the overall change in the game's potential. From Eqs. 3.20, 3.21, and 3.24, the potential of edge server $a_i'$ increases by,

$$\begin{aligned}\Phi_{inc} &= (\Phi_{\zeta_{a_i'}}(R^a_{a_i'} \cup \{i\}, \zeta_{a_i'}) + \Phi_{q_{a_i'}}(R^a_{a_i'} \cup \{i\}, q_{a_i'})) - (\Phi_{\zeta_{a_i'}}(R^a_{a_i'}, \zeta_{a_i'}) + \Phi_{q_{a_i'}}(R^a_{a_i'}, q_{a_i'})) \\ &= (\Phi_{\zeta_{a_i'}}(R^a_{a_i'} \cup \{i\}, \zeta_{a_i'}) - \Phi_{\zeta_{a_i'}}(R^a_{a_i'}, \zeta_{a_i'})) + (\Phi_{q_{a_i'}}(R^a_{a_i'} \cup \{i\}, q_{a_i'}) - \Phi_{q_{a_i'}}(R^a_{a_i'}, q_{a_i'})).\end{aligned} \tag{3.28}$$

From Eq. 3.22 and Eq. 3.24,

$$\Phi_{inc} = \varrho_i(a_i', a_{-i}) + q_{a_i'}(R^a_{a_i'} \cup \{i\}). \tag{3.29}$$

Similarly, the potential of edge sever $a_i$ decreases by,

$$\begin{aligned}\Phi_{dec} &= (\Phi_{\zeta_{a_i}}(R^a_{a_i}, \zeta_{a_i}) + \Phi_{q_{a_i}}(R^a_{a_i}, q_{a_i})) - (\Phi_{\zeta_{a_i}}(R^a_{a_i} \setminus \{i\}, \zeta_{a_i}) + \Phi_{q_{a_i}}(R^a_{a_i} \setminus \{i\}, q_{a_i})) \\ &= \varrho_i(a_i, a_{-i}) + q_{a_i}(R^a_{a_i})\end{aligned} \tag{3.30}$$

From Eqs. 3.26 and 3.27, $\Delta\Phi = \Delta c_i < 0$. Hence, the UAGame is the potential game. It also implies that $\Phi$'s value is decreased when users update their allocation decisions under the AUA algorithm. □

**Theorem 3.2** *The UAGame admits at least one PNE under the AUA algorithm.*

**Proof:** The PNE existence can be proved by using following arguments:

1. In the UAGame, potential function $\Phi(a)$'s values are mapped with allocation profiles, and the number of allocation profiles is finite. As a result, $\Phi(a)$'s outcomes will be finite.
2. UAGame transits from one state to another in terms of allocation profiles as app users update their allocation decisions. The potential function $\Phi(a)$'s value monotonically decreases with every allocation decision update by app users under the AUA algorithm, as proved in Theorem 3.1. Consequently, the states of the UAGame will not repeat.

The results from the above arguments mean that, eventually, the AUA algorithm stops. It means that no app user has the option of deviating from their existing allocation decisions, which is a necessary condition of PNE. Hence, UAGame admits at least one PNE. □

***Complexity Analysis*:** It is also possible to determine how many iterations are required to achieve a PNE using the above findings. As aforementioned, $\Phi(a)$ is used to track the AUA algorithm's convergence process. In UAGame, if all users in the coverage area of server $x$ are allocated to it, the value of $\Phi_x(a)$ (defined in Eq. 3.20) will be maximized. On the other hand, the minimum potential value of an edge server $x$ will be if no app users are allocated to it. Furthermore, as shown in Theorem 3.1, the overall UAGame potential $\Phi(a)$ decreases as allocation decisions are updated by users using the AUA algorithm. Theorem 3.2 states that the AUA algorithm completes the overall system cost and QoS optimization process without a cycle and reaches a PNE in finite iterations. The AUA algorithm takes $M$ groups of edge servers as an input. Users can make parallel decisions on all the servers in each group $T_l \subseteq S$ during each algorithm iteration. Let an edge server $x$ covers $n_x$ app users.The above findings imply that the maximum iterations required to reach PNE is $\max_{n_x}(M \times {n_x}^2)$, where $n_x \ll n$. Only the app users, in the coverage area of more than one edge server, have the option to change their allocation decisions. If half of the nodes are in the shared area, the time

complexity to reach PNE will be $O(\max_{n_x}(M \times (\frac{n_x}{2})^2))$. Hence, in the real edge server network with large geographical area, the time complexity to reach PNE will not be too high.

## 3.7 Theoretical Evaluation

The AUA algorithm proposed a solution known as PNE, which minimizes each app user's cost function (including usage and QoS-degradation costs), as discussed in Section 3.3.6. This section examines the optimality achieved by the AUA algorithm in the worst-case scenario when each app user offloads a task of maximum size. In the worst case, an application's most expensive computational tasks that could be offloaded are similar in size. Furthermore, in most apps such as natural language processing, face recognition, security management apps, traffic management apps, environmental monitoring apps, etc., each device offloads a similar-sized computational task to the edge server [41]. In such cases, the Shapley value concepts assign an equal portion of the edge server's cost to each app user as discussed in Section 3.3.4. Thus, the usage cost incurred by each user $i$ is calculated from Eq. 3.8 as follows:

$$\varrho_i(a) = \frac{\zeta_x(R_x^a)}{|R_x^a|}. \tag{3.31}$$

**App users' overall cost** The sum of all servers' costs (including both usage and QoS-degradation costs) is the total cost of app users. Given edge server set $S$, app user set $U$, and an allocation profile $a$, the total cost (denoted by $C(a)$) of app users can be calculated from Eq.3.11 as follows:

$$C(a) = \sum_{x \in S} \zeta_x(R_x^a) + |R_x^a| \cdot q_x(R_x^a) = C_\zeta(a) + C_q(a), \tag{3.32}$$

where $C_\zeta(a) = \sum_{x \in S} \zeta_x(R_x^a)$ and $C_q(a) = \sum_{x \in S} |R_x^a| \cdot q_x(R_x^a)$ are the app users' total usage and QoS-degradation cost, respectively.

**Overall potential $\Phi(a)$ in the worst case** In this case, the value of potential function from Eq. 3.25 at an allocation profile $a$ will be as follows,

$$\Phi(a) = \sum_{x\in S}\sum_{k=1}^{|R_x^a|}\left(\frac{\zeta_x(k)}{k} + q_x(k)\right) = \Phi_\zeta(a) + \Phi_q(a), \tag{3.33}$$

where $\Phi_\zeta(a) = \sum_{x\in S}\sum_{k=1}^{|R_x^a|}\frac{\zeta_x(k)}{k}$ and $\Phi_q(a) = \sum_{x\in S}\sum_{k=1}^{|R_x^a|} q_x(k)$. Here $\Phi_\zeta(a)$ and $\Phi_q(a)$ are the overall potentials of UAGame due to usage and QoS-degradation cost, respectively. The potential function can be written as follows,

$$\Phi(a) = \sum_{x\in S}\left(\zeta_x(k)\cdot H(|R_x^a|) + \sum_{k=1}^{|R_x^a|} q_x(k)\right), \tag{3.34}$$

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

**Price of Stability** Price stability (PoS) is a general indicator of the quality of a PNE. The PoS is a ratio between total cost $C(a)$ (Eq. 3.32) at the best PNE and the centralized optimum, where the best PNE is an stable optimum. If centralized and PNE solutions are allocation profiles $a^o$ and $a^*$, respectively, the PoS is,

$$pos(a) = \frac{C(a^*)}{C(a^o)}. \tag{3.35}$$

We establish a bound on the PoS in the worst-case where each app user offloads a task of maximum size: edge servers attain the maximum workload. The UAGame's potential is crucial in determining the PoS bound. Therefore, a relationship between the app users' total usage cost $C_\zeta(a)$ and the UAGame's potential $\Phi_\zeta(a)$ due to the usage cost is established, as follows in Lemma 3.1.

**Lemma 3.1** *If each edge server $x$ has a non-decreasing concave cost function $\zeta(.)$, then $C_\zeta(a) \leq \Phi_\zeta(a) \leq C_\zeta(a)\cdot H(n)$ at any allocation profile $a$, where $H(n) = \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$.*

**Proof:** Let see the first inequality $C_\zeta(a) \leq \Phi_\zeta(a)$, where $C_\zeta(a) = \sum_{x\in S}\zeta_x(R_x^a) =$

$\sum_{x\in S} |R_x^a| \cdot \frac{\zeta_x(R_x^a)}{|R_x^a|}$, and $\Phi_\zeta(a) = \sum_{x\in S}\sum_{k=1}^{|R_x^a|} \frac{\zeta_x(k)}{k}$. The function $\zeta_x(.)$ is a concave, so the cost per app user $\frac{\zeta_x(R_x^a)}{|R_x^a|}$ will decrease with the increasing number of app users $|R_x^a|$. Therefore, $C_\zeta(a) \leq \Phi_\zeta(a)$.

The second inequality $\Phi_\zeta(a) \leq C_\zeta(a) \cdot H(n)$ can be written as $\sum_{x\in S}\sum_{k=1}^{|R_x^a|} \frac{\zeta_x(k)}{k} \leq \sum_{x\in S} H(n) \cdot \zeta_x(R_x^a)$. Since $\zeta_x(R_x^a)$ is a non-decreasing function, so inequality $\Phi_\zeta(a) \leq C_\zeta(a) \cdot H(n)$ follows. □

**Theorem 3.3** *In UAGame, if $\zeta_x(.)$ is a non-decreasing concave function, $q_x(.)$ is a non-decreasing function for all $x$, and $C_q(a) \leq \alpha \cdot \Phi_q(a)$, then the PoS never exceeds $\alpha \cdot H(n)$.*

**Proof:** Assume that $C(a^o)$ represents the total cost of app users at the centralized optimal solution $a^o$. Let users start with allocation profile $a^0$ and then change their allocation decisions using the AUA algorithm to reach PNE solution $a^*$. The function Φ(a) decreases with each AUA algorithm's iteration as stated in Theorem 3.1, so,

$$\Phi(a^*) \leq \Phi(a^o) \tag{3.36}$$

From the first inequality of Lemma 3.1, $C_\zeta(a^*) \leq \Phi_\zeta(a^*)$ and assumption, $C_q(a^*) \leq \alpha \cdot \Phi_q(a^*)$, we can say that $C_\zeta(a^*) + C_q(a^*) \leq \Phi_\zeta(a^*) + \alpha \cdot \Phi_q(a^*)$. From the Eq. 3.36, $C_\zeta(a^*) + C_q(a^*) \leq \Phi_\zeta(a^o) + \alpha \cdot \Phi_q(a^o)$. The second inequality of Lemma 3.1 gives $\Phi_\zeta(a^o) \leq C_\zeta(a^o) \cdot H(n)$, so $C_\zeta(a^*) + C_q(a^*) \leq H(n) \cdot C_\zeta(a^o) + \alpha \cdot \Phi_q(a^o)$. After expending the right hand side of this inequality from Eq. 3.18 and Eq. 3.33, we get,

$$C_\zeta(a^*) + C_q(a^*) \leq \sum_{y\in S}\left(H(n) \cdot \zeta_y(R_y^{a^o}) + \alpha \cdot \sum_{k=1}^{|R_y^{a^o}|} q_y(k)\right).$$

Here, $q_y(.)$ is a non-decreasing function, so $\sum_{k=1}^{|R_y^{a^o}|} q_y(k) \leq |R_y^{a^o}| \cdot q_y(R_y^{a^o})$. Thus, inequality further can be written as,

$$C_\zeta(a^*) + C_q(a^*) \leq \sum_{y \in S} H(n) \cdot \zeta_y(R_y^{a^o}) + \alpha \cdot |R_y^{a^o}| \cdot q_y(R_y^{a^o}).$$

Both $\zeta(.)$ and $q(.)$ are non-negative, and $\sum_{y \in S} \zeta_y(R_y^{a^o}) > 0$ and $\sum_{y \in S} |R_y^{a^o}| \cdot q_y(R_y^{a^o}) > 0$, so,

$$\begin{aligned} C_\zeta(a^*) + C_q(a^*) \quad &\leq \quad H(n).\alpha \sum_{y \in S} (\zeta_y(R_y^{a^o}) + |R_y^{a^o}| \cdot q_y(R_y^{a^o})) \\ &\Rightarrow \quad C(a^*) \leq H(n) \cdot \alpha \cdot C(a^o). \end{aligned} \tag{3.37}$$

Hence, the PoS is at most $\alpha \cdot H(n)$. □

## 3.8 Experiment and Analysis

This section examines the AUA algorithm's performance in terms of different matrices by performing vast experiments with the diverse geographical densities of users and servers.

### 3.8.1 Performance Benchmark

Our proposed algorithm is compared to four state-of-the-art approaches to solve the AUA problem.

- TPDS19 [41]: It proposes a game-theoretic method for solving the AUA problem that aims to optimize the cost and the number of served users.
- MCFH20 [42]: It solves the AUA problem by heuristically assigning the edge server with the most capacity first, and optimizes the overall cost and the number of served users.
- NOMA-EUA21 [79]: This study presents a game-theoretic method that allocates users to the servers in order to maximize the vendor's benefits.
- OSRA22 [128]: This study proposes a decentralized approach for edge resource allocation that aims to optimizes the services cost.

**Table 3.1**: Parameters' Setting

| Name | Description |
|---|---|
| Preference weight $h_d$ for each $d \in D$ | 0.25 |
| QoS weight $I_\lambda$ for each $\lambda \in \Pi$ | 0.25 |
| Tunable parameter $\alpha_\lambda$ for each $\lambda \in \Pi$ | 0.4 |
| Tunable parameter $\gamma_\lambda$ for each $\lambda \in \Pi$ | 2 |
| Parameter $\theta_d^\lambda$ | 0.25 |
| CPU Scheduling Algorithm | RR Algorithm |
| CPU Burst time for 1 Kb size task | 1 ms |
| Time quantum | 10 ms |
| Process transition time on CPU | 2 ms |
| Waiting queue size for each resource | 5 |
| memory access time | 20 ms |

**Table 3.2**: Experimental Settings

| | Number of app users | Number of edge servers | Edge server Range | Task size |
|---|---|---|---|---|
| set#1 | 100,...,1000 | 50% | [750m, 1500m] | [15$kb$, 40$kb$] |
| set#2 | 500 | 10%,...,100% | [750m, 1500m] | [15$kb$, 40$kb$] |

The experiments are written in Python 3.5 and run on a Windows 10 computer with an Intel CORE i7-7700U processor running at 8 GB of RAM and 3.60 GHz.

### 3.8.2 Simulation Settings

For analyzing the AUA algorithm, a rectangular space of 20 km x 20 km is created. In this rectangular space, the base stations are placed 600 m away from each other. We randomly placed 120 edge servers at different base stations for each simulation. The coverage radius of edge servers is uniformly distributed over [750 m, 1500 m]. Normal distribution $N(\mu, \sigma^2)$ generates the available resources for each server randomly, where $\sigma = 3$ is the standard deviation, and $\mu = 10$ is the average amount of each resource type. For a resource type, a negative quantity may be produced under the normal distribution that is rounded to one. Each edge server has four type of resources as {*memory, bandwidth, CPU, Storage*}. Users generate offloaded tasks of varying sizes

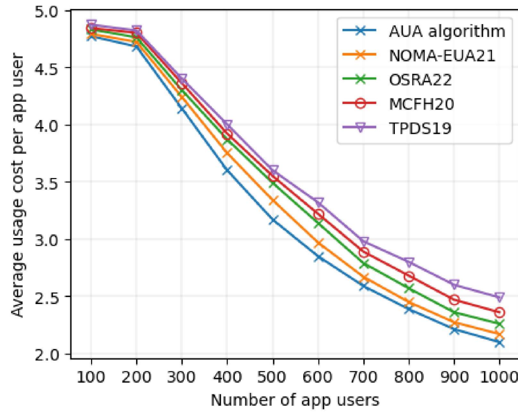
by uniform distribution from [15 $Kb$, 40 $Kb$]. In our experiments, the parameter $y_d$ ($0.9 < y_d < 1$) is calculated by the average task size. The required resources for running the task will be decided at run time based on the task size. Each user generates the computational task randomly from a uniform time interval distribution [1 s, 5 s]. Table 3.1 summarizes the other parameters that are used. Each experiment is repeated 50 times, and the results are averaged to eliminate extreme possibilities such as dense or sparse server/user distributions. Experiments 1 and 2 correspond to data sets 1 and 2, which are summarized in Table 3.2.

### 3.8.3 Performance Analysis

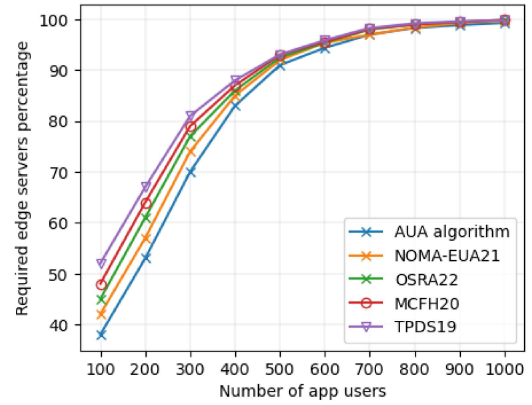
We examine usage cost, required edge servers, the number of allocated users per server (resource utilization), total allocated users (resource availability), resource wait time, and QoS-degradation cost to determine the algorithm's performance in achieving the optimization objective.

#### 3.8.3.1 Experiment 1

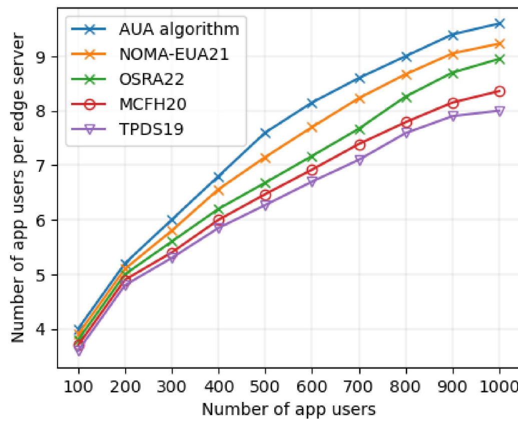
Fig. 3.4 depicts the experimental findings that assess the algorithm's performance with varying user numbers from 100 to 1000, where edge servers are fixed at 60. Fig. 3.4a illustrates that when the number of users grows, each user's usage cost decreases as resource sharing increases due to multi-tenancy. When the users increase from 100 to 200, the usage cost does not decrease significantly because resource sharing occurs on a smaller scale. The AUA algorithm assigns users to an edge server until their QoS is acceptable. As a result, the AUA algorithm effectively employs the multi-tenancy architecture, making it more cost-effective in terms of usage. The required edge server numbers grow directly proportional to the number of users, as illustrated in Fig. 3.4b. This happens because load balancing is used to maintain QoS as users increase, which increases the number of servers in use. Thus, at some users, all approaches utilize the



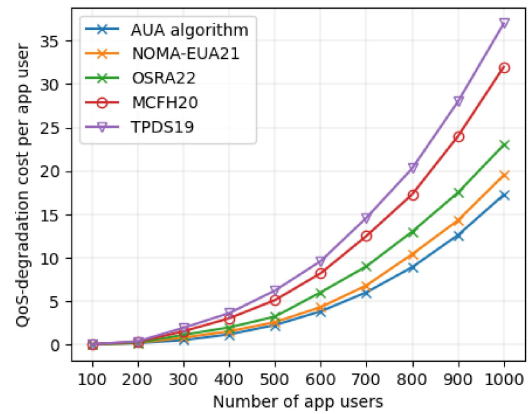

(a) Set#1: Users vs. usage cost.



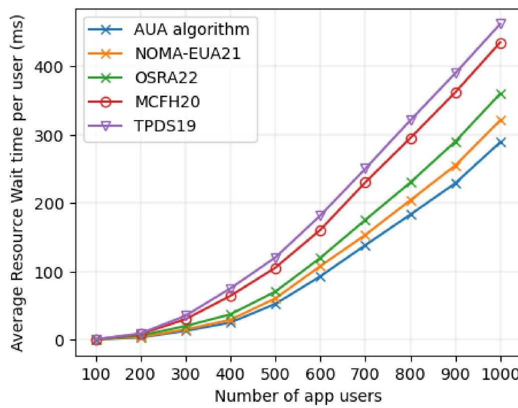

(b) Set#1: Users vs. required servers.



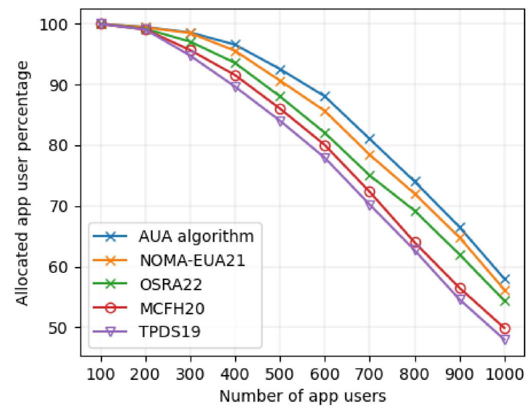

(c) Set#1: Users vs. users per server.




(d) Set#1: Users vs. QoS-degradation cost.

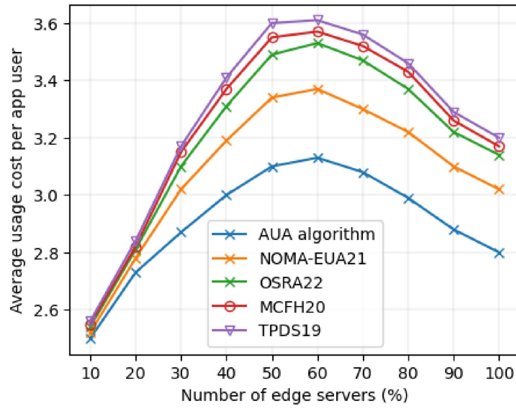



(e) Set#1: Users vs. resources wait time.

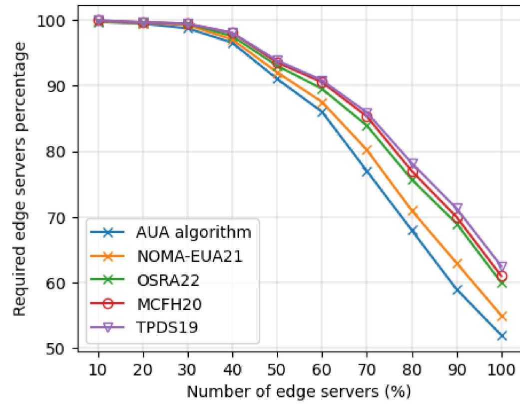



(f) Set#1: Users vs. total allocated users.

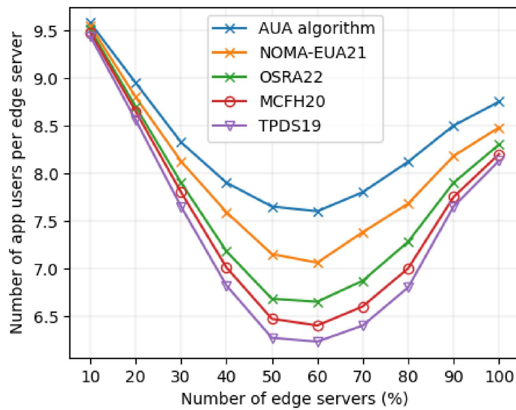
**Figure 3.4**: Experimental results show the AUA algorithm's performance with varying numbers of app users.



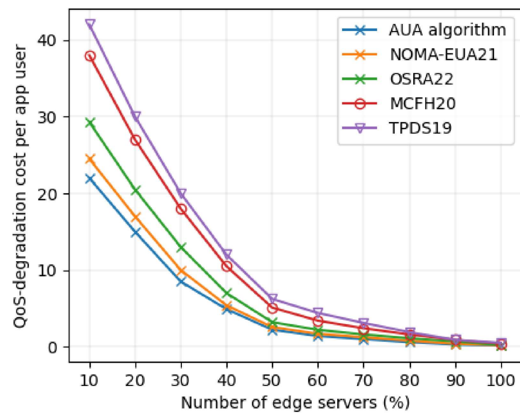

(a) Set#2: Servers vs. usage cost.



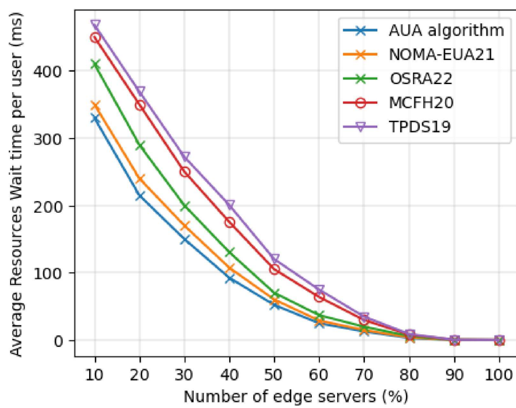

(b) Set#2: Servers vs. required servers.



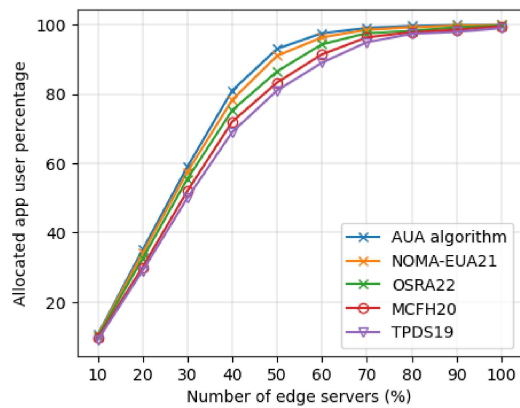

(c) Set#2: Servers vs. users per server.




(d) Set#2: Servers vs. QoS-degradation cost.




(e) Set#2: Servers vs. resources wait time.

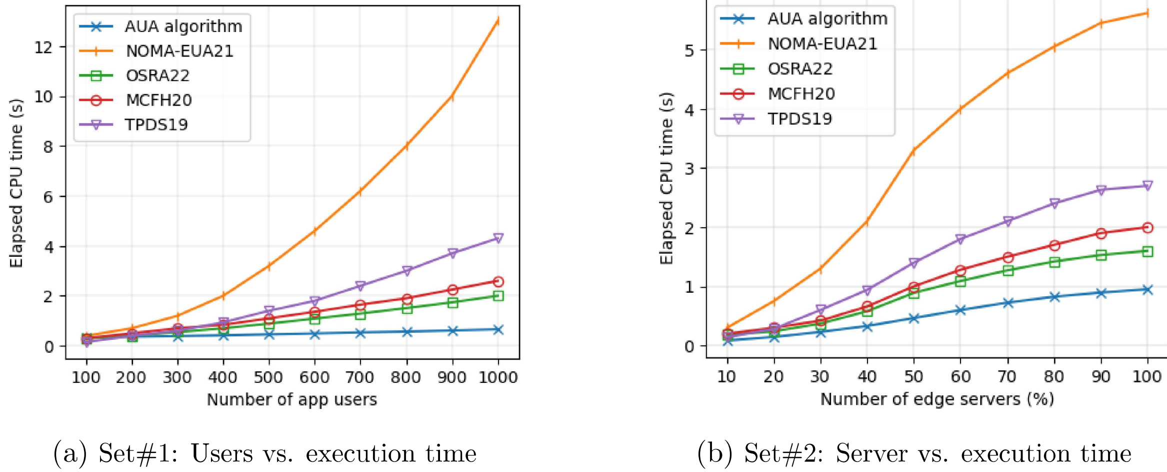



(f) Set#2: Servers vs. total allocated users.

**Figure 3.5**: Experimental results show the AUA algorithm's performance with varying numbers of edge servers.

most available edge servers, such as 700, in our tests. These findings validate that our proposed approach uses servers more efficiently than others, lowering overall usage costs. The OSRA22, NOMA-EUA21, MCFH20, and TPDS19 approaches do not perform well because they do not balance the load on the edge servers, resulting in some edge servers being overloaded and others being underutilized (having very few users). The AUA, on the other hand, allocates app users in such a way that the load on the edge servers is balanced, optimizing the required edge servers. Fig. 3.4c depicts the average number of users assigned to servers. In other words, it shows the resource utilization on the edge servers. Users on given servers will naturally increase in tandem with increasing user numbers in the system. The AUA algorithm reduces usage costs by maximizing the use of multi-tenant servers and minimizing their numbers in use. Thus, the AUA algorithm effectively implements the multi-tenancy model, giving it an advantage over other algorithms.

Fig. 3.4d shows that the QoS-degradation cost increases with increasing users allocated to servers. It is because if the number of users on an edge server exceeds a specific limit, the QoS for users allocated to that edge server will be degraded. The QoS-degradation cost starts to rise after some point, e.g., it starts to increase after 200 users in our experiment since most of the edge servers were underutilized when users were less than 200. The OSRA22, TPDS19, and MCFH20 do not balance the load on the edge servers and leave more users unallocated, resulting in poor QoS performance. The NOMA-EUA21 method performs well for a small number of users. Still, it is not practical for a considerable number of users as the time complexity grows exponentially. On the other hand, the AUA algorithm establish a better equilibrium between usage cost and QoS, and it allocates users to servers without degrading the required QoS. Likewise, the waiting time of each user increases as the resource contention rises with increasing user numbers, as depicted in Fig. 3.4e. The proposed AUA algorithm performs better as it raises the number of assigned users to any server only as long as




(a) Set#1: Users vs. execution time

(b) Set#2: Server vs. execution time

**Figure 3.6**: Results show the CPU time required to conduct Experiments 1 and 2.

the utilization of resources on the server increases. Fig. 3.4f shows the allocated user percentage to servers. The resource availability decreases with an increasing number of users, leading to the allocation percentage decreasing. The AUA performed better than others as it effectively models the multi-tenancy architecture while maintaining the required QoS for users.

#### 3.8.3.2 Experiments 2

In this experiment, we analyze the AUA algorithm by varying the edge servers from 10% to 100% of total edge servers (120) and users is fixed at 500. The edge servers vary in steps of 10%, (Fig. 3.5). The findings in Fig. 3.5a show the variation in the average usage cost of each allocated user while increasing edge servers. At the beginning point (10% edge servers), multi-tenancy is very high as fewer servers are available than required. Consequently, the allocated users' usage cost is less. Conversely, Fig. 3.5d shows that the QoS-degradation cost is very high at this point due to the resource unavailability and resource sharing overhead. As the available servers increase, so does the number of users getting served, as shown in Fig. 3.5f. Furthermore, increasing the edge servers also reduced the resource sharing overhead. Because more users are served and resource sharing overhead is reduced, the cost of QoS degradation declines as a

result, as shown in Fig. 3.5d. On the other hand, Fig. 3.5a illustrates that each user's usage cost increases as the multi-tenancy decreases. However, the average usage cost begins to decrease after a point, such as 60% in our experiment. As the percentage of edge servers reaches 60%, more options for allocating users become available; some may find a server with greater multi-tenancy benefits in their coverage area, allowing them to reduce their usage costs. The proposed AUA algorithm outperforms the OSRA22, NOMA-EUA21, MCFH20, and TPDS19 as it does not allow user allocation to the overloaded edge servers.

Figure Fig. 3.5b shows the needed edge servers: after a point where available edge servers are sufficient, the required edge server percentage in all approaches rapidly decreases. It demonstrates that how these approaches are effective in resource utilization. The AUA outperforms OSRA22, NOMA-EUA21, MCFH20, and TPDS19 because it better utilizes the resources on the multi-tenant edge servers. The MCFH20, and TPDS19 use more edge servers as these approaches allocate the users to minimize the cost without considering the multi-tenancy. Similarly, when it comes to the number of users allocated to each server, the AUA outperforms others and makes better use of resources by allocating more users to each required edge server, as shown in Fig. 3.5c. The variation in the number of users per server follows the opposite behavior of usage cost: more users per server results in less cost to each user. The results in Fig. 3.5e show that the average resource waiting time decreases as more servers are available. The waiting time of resources follows a similar pattern as QoS-degradation cost.

### 3.8.4 Efficiency

Fig. 3.6 shows the algorithm's efficiency in terms of average CPU execution time to allocate the users to the servers in Experiments 1 and 2. The results in Fig. 3.6a show that the CPU time taken by NOMA-EUA21 and TPDS19 increases rapidly with the increasing number of users as it allows users one by one to make decisions. In the worst

case, NOMA-EUA21 takes exponential time to reach the PNE. The execution time of MCFH20 and OSRA22 to solve the problem also increases since heuristic approaches take a higher execution time to get the effective solution with a large number of users. Given an edge server set, the maximum number of users in the coverage range of any server, rather than the total number, determines the AUA algorithm's time complexity as it divides the network into groups and solves the problem in parallel. The increase rate of users per server is much slower than the overall user increase rate in the system. Thus, the increment in CPU time taken by the proposed AUA algorithm is lower.

The results in Fig. 3.6b show that the CPU time taken by various approaches is increased since the network stretched with an increasing number of edge servers. Given a user set, the AUA algorithm's time complexity depends on the number of groups determined by the number of colors used in distance-2 graph coloring. The increase in the number of groups is much slower than the increase in the total number of edge servers. As a result, the AUA algorithm's convergence time is lower than others. However, as the number of edge servers increases, so does the CPU time required for the grouping process (Fig. 3.6b).

## 3.9 Summary

This chapter proposed a protocol to establish the trade-off between the usage cost and the Quality of Service (QoS) while balancing the load on edge servers. In this study, we formulated the App User Allocation (AUA) problem as a UAGame, a potential game that admits at least one Pure Nash Equilibrium (PNE). An App User Allocation (AUA) algorithm that runs in parallel on the edge servers is designed to find the PNE and converge quickly. The time complexity of AUA algorithm has a bound $O(\max_{n_x}(M \times (\frac{n_x}{2})^2))$ for convergence to PNE. We also theoretically analyzed the solution's optimality in terms of Price of Stability (PoS) and found the bound for PoS that is at most $\alpha \cdot H(n)$. The simulation also depicted that the AUA algorithm minimizes the overall cost and

improves the QoS compared to other state-of-the-art approaches.