Chapter 5

Allocating Resources with Variable Coverage of Servers

In the previous chapters, we investigated the problem of edge computing resource allocation from the perspective of the app users and vendors. Primarily, these chapters focused on improving the QoS and optimizing the cost. In this chapter, we investigate the problem from the perspective of edge infrastructure and its users. The proposed approach in this chapter accommodates the geographically dynamic user density by moving the users from the overloaded edge servers to underutilized edge servers.

5.1 Introduction

Generally, different geographical areas have different user densities [134, 135]. For example, the working areas, such as the offices, markets, or industrial premises, have high user density in the daytime. On the other hand, the residential areas follow the opposite pattern. In the higher user density areas, some edge servers may have allocated to more users than their capacities, leading to an overloaded condition of edge servers. This situation also causes higher waiting times for resources along with resource sharing overhead, resulting in a lower Quality of Service (QoS) provided to the users. Further-

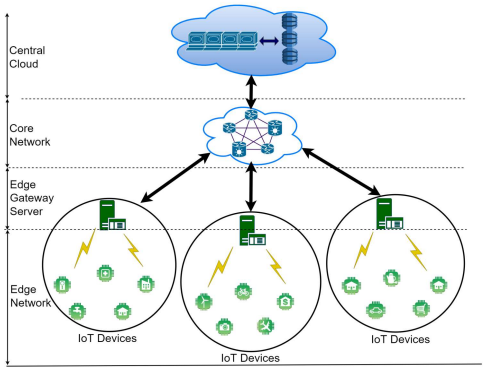


Figure 5.1: IoT devices and environments with supporting computing paradigms.

more, it affects users' perceived Quality of Experience (QoE) because some users may not be assigned to any of the edge servers due to overloaded edge servers. On the other hand, some edge servers may be underutilized in low user density areas, resulting in lower resource utilization. Thus, the geographically dynamic user density results in uneven loads on different edge servers. Hence, an appropriate user allocation is required for balancing the load on edge servers to maximizes their utilization and providing a better QoS and QoE to users. This user allocation problem is referred to as the *Edge Resource Allocation* (ERA) problem.

In the literature, by considering each user as a tenant, edge servers are classified into two categories: single-tenant and multi-tenant. In case of multi-tenancy, a single instance of software running on a supporting infrastructure serves multiple users, whereas only one user in single-tenancy. Unlike the central cloud [136], edge servers have limited resources in terms of memory, storage, CPU, bandwidth, etc. The utilization of each type of edge resource is influenced differently by the various types of services offered by the application providers [41]. For example, complex computational-based applications demand more CPU resources, whereas multi-media streaming-based applications demand more bandwidth. Therefore, a few resources may fall short on any edge server, whereas others may be adequate. The resources that are falling short are the bottleneck and result in overloading. Thus, the edge server's performance depends on the bottleneck resource.

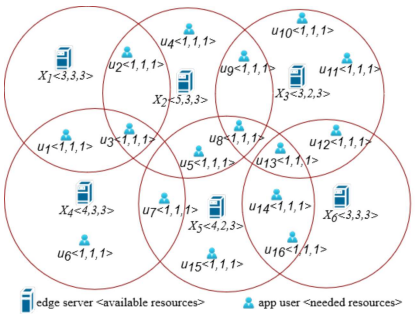


Figure 5.2: The example of user allocation to edge servers.

This chapter proposes a distributed approach to solve the ERA problem that accommodates the geographically dynamic user density while dealing with the bottleneck resources. The key is to vary the extent of the geographic area covered by the underutilized edge servers to accommodate users from overloaded edge servers. The other primary objective of our proposed approach is to maximize the resource utilization of edge servers by efficiently employing multi-tenancy. As a result, it improves the QoS and QoE for users. The main contributions of this chapter are as follows:

- Each resource type's utilization on each multi-tenant edge server is formulated in terms of the benefits function of edge servers.
- A critical point is evaluated for the benefit function of each resource type. It is proven that the value of the benefit function is maximum at this critical point.
- With the help of maxima points, the bottleneck computing resources on the multi-tenant edge servers are identified and the capacity constraint of edge servers is defined.
- Based on the above findings, the ERA problem is formulated as a constrained optimization problem in terms of the edge servers' benefit functions.
- An ERA Algorithm is employed to reallocate users from overloaded to underutilized edge servers.
- The performance of the ERA algorithm in terms of resource utilization, availability, and latency is numerically compared to five cutting-edge approaches.

The remaining chapter is structured as follows. Section 5.2 focuses on the system paradigm and Section 5.3 formulates the ERA problem. The ERA algorithm is presented in Section 5.4. The performance of the ERA algorithm is examined in Section 5.5.

5.2 System model

To investigate the problem of edge resource allocation, an edge computing environment is considered where m multi-tenant edge servers $S = \{1, 2, 3, \cdots, m\}$ are deployed at different locations. These edge servers provides their services to n users $U = \{1, 2, 3, \cdots, n\}$. An edge server x's computing resources are represented by a vector $W_x = (W_x^d)$, where $d \in D = \{\text{CPU, bandwidth, memory, storage} \cdots\}$ refers to the different types of computing resources.

Definition 5.1 (Proximity constraints) *A user i can use the services of an edge server x only if user i present in the coverage range of x as follows:*

$$i \in cov(x),\ \forall i \in U,\ \forall x \in S, \tag{5.1}$$

where $cov(x)$ denotes the coverage range of edeg server x.

Any user can be present in multiple edge servers' coverage areas, and one of the edge servers is assigned to that user. Edge server set A_i ($A_i \subseteq S$) denotes the set of all edge servers covering a user i.

Definition 5.2 (Server Allotment Decision) *Given edge server set $S = \{1, \cdots m\}$ and a user i, a server allotment decision specifies an edge server that is assigned to user i. It is denoted by a_i, where $a_i \in A_i$, here A_i is a server allotment decision set.*

Definition 5.3 (Server Allotment Profile) *Given edge server set $S = \{1, \cdots m\}$ and user set $U = \{1, 2, 3, \cdots, n\}$, a server allotment profile is a tuple of simultaneous*

server allotment decisions for all users. It is indicated by $a = (a_1, a_2, \cdots a_n)$*, where each* $a_i \in A_i$ *is a server allotment decision of user* i*.*

The cartesian product of all users' server allotment decision sets, $A = A_1 \times A_2 \times A_3 \times \cdots \times A_n$, is the set of all possible server allotment profiles denoted by A.

5.2.1 Edge Server revenue model

Edge servers generate revenue by renting out computing resources. This chapter investigates the pay-as-you-go pricing model, which determines generated revenue in proportion to resource utilization. Eq. 3.5 in Chapter 3 calculates the utilization of each type of resource on an edge server. If the total available resource on edge server x is $W_x = (W_x^d)$, the revenue generated by x is calculated by aggregating the revenue generated by all the units of each resource type:

$$\zeta_x(R_x^a) = \sum_{d \in D} h_d \cdot Z_d(x) \cdot W_x^d = \sum_{d \in D} -h_d \cdot log_{\gamma_d}(\frac{|R_x^a| \cdot \delta_d}{W_x^d}) \cdot W_x^d, \tag{5.2}$$

where, h_d is the weight assigned to resource type d, such as $\sum_{d \in D} h_d = 1$.

5.2.2 Overhead Cost Model

User queuing time, process transition overhead, channel traffic, etc., all increase as the users increase on an edge server. At some point, when the number of users exceeds a certain threshold, a significant amount of time is spent managing the resource overhead, which cuts the valuable time to process the users' offloaded tasks. In this state, the edge server is referred to as an *overloaded server*, and its performance suffers. This decrease in performance is viewed as an overhead cost by edge servers, which is calculated as follows:

$$Q_x(R_x^a) = \sum_{d \in D} \theta_d \cdot G_d(R_x^a) \tag{5.3}$$

Where θ_d is the assigned weight to resource type d, such as $\sum_{d \in D} \theta_d = 1$, and $G_d(\cdot)$ is a normalized function. The normalized function defines the trajectories of the different resource type. The most common normalization functions are the logarithm form [137], the exponential form [19,138], the linear piece-wise form [139], the sigmoid forms [140]. This chapter employs the exponential form, which rapidly increases as users exceed a threshold. The normalized function's exponential form is as follows:

$$G_d(R_x^a) = e^{-\alpha_d \cdot Z_d(x) - \beta_d} = e^{-\alpha_d \cdot log_{\gamma_d}(\frac{|R_x^a| \cdot \delta_d}{W_x^d}) - \beta_d}, \tag{5.4}$$

where $0 < \alpha_d < 1$ is a tunable parameter that controls the function's growth rate, β_d controls at what point the function's growth should begin. Here, normalized function increases with the increase in the number of users.

5.2.3 Edge Server Benefit Model

A lower $\zeta_x(R_x^a)$ value specifies that edge server x is underutilized, while a higher $Q_x(R_x^a)$ value indicates the overloading of edge server x. Thus, in order to improve edge server x's service performance and resource usage, the difference term $(\zeta_x(R_x^a) - Q_x(R_x^a))$ must be maximized. Term $(\zeta_x(R_x^a) - Q_x(R_x^a))$ is referred to as the net benefit of a server x, which is specified by Y_x as:

$$Y_x(R_x^a) = \zeta_x(R_x^a) - q_x(R_x^a). \tag{5.5}$$

If a server is underutilized, its users must be increased, and vice versa if it is overloaded. Hence, the number of users in set R_x^a assigned to edge server x should be chosen in such a way that Y_x is maximized:

$$\max_{R_x^a} \; Y_x(R_x^a). \tag{5.6}$$

The maximization of $Y_x(\cdot)$ increases the valuable time used to process off-loaded tasks. As a result, user QoS is improved while resource utilization is optimized.

5.2.4 Coverage Range of Edge Servers

Edge infrastructure providers define the coverage area range of each edge server during deployment based on different factors, e.g., user density, energy consumption, non-service area, etc. The coverage area of an edge server can be represented in terms of the radius of the coverage area. In this study, edge servers' coverage area varies based on user density between a minimum and maximum coverage area. The minimum coverage range of edge servers is defined to avoid the non-service areas (not covered by any edge server). Tuple $(r^1_{min}, r^2_{min}, \cdots, r^m_{min})$ denotes the minimum coverage area of edge servers, where each r^x_{min} denotes the minimum coverage area of server x. The coverage area of an edge server may exceed its minimum coverage area according to geographical user density. However, it should not exceed a certain point that increases the energy consumption of users' devices. This point is the maximum coverage range r^x_{max} of an edge server, defined using the First Order Radio Energy Model.

5.3 Properties and Problem Formulation

5.3.1 System Properties

This subsection defines the bottleneck resources and the capacity constraint for each edge server. For this purpose, the benefits gained from each individual resource type on each edge server are used. The benefit from resource type d on edge server x, specified by $Y_d(R^a_x)$, can be calculated from Eqs. 5.2 and 5.3 as follows:

$$Y_d(R^a_x) = -h_d \cdot log_{\gamma_d}(\frac{|R^a_x| \cdot \delta_d}{W^d_x}).W^d_x - \theta_d \cdot e^{-\alpha_d \cdot log_{\gamma_d}(\frac{|R^a_x| \cdot \delta_d}{W^d_x}) - \beta_d}. \tag{5.7}$$

The trajectory of the benefit function $Y_d(R^a_x)$ for any multi-tenant resource type d is shown in Fig. 5.3. Initially, the profit function grows with number of users because the

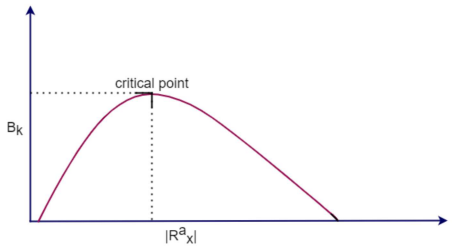


Figure 5.3: The trajectory of benefit function for any resource type with the number of users.

edge server is underutilized and the increase in the computational tasks' execution time is more than the growth in the time spent managing resource overhead. At some point, it starts to decrease as time spent managing resource overhead exceeds the computational tasks' execution time. This condition indicates the edge server is overloading.

We find the critical point of the benefit function in terms of the number of allocated users $|R_x^a|$ to each resource type d of the edge server x and analyze the benefits of the edge server for each resource type at the critical point. The critical points of resource types identify the bottleneck resources on each edge server.

Lemma 5.1 *Given benefit function $Y_d(R_x^a)$ for a resource type d on edge server x, the critical point is $\frac{W_x^d}{\delta_d \cdot \tau^{ln(\gamma_d)} \cdot \gamma_d^{\beta_d}}$, where $\tau = \frac{h_d \cdot W_x^d}{\theta_d \cdot \alpha_d}$.*

Proof: The critical point specifies how many users $|R_x^a|$ will be allocated to resource type $d \in D$ of the edge server x at the maximum or minimum value of benefit function $Y_d(R_x^a)$, where R_x^a and $|R_x^a|$ are the set of users allocated to edge server x and the cardinality of R_x^a, respectively. To find the cardinality of R_x^a as a critical point, we evaluate the *first derivative* of benefit function $Y_d(R_x^a)$ that is as follows:

$$\frac{d}{d(|R_x^a|)} \cdot Y_d(R_x^a) = \frac{-h_d \cdot W_x^d}{R_x^a \cdot ln(\gamma_d)} + \frac{\theta_d . \alpha_d}{R_x^a \cdot ln(\gamma_d)} \cdot e^{-(\alpha_d \cdot log_{\gamma_d}(\frac{R_x^a \cdot \delta_d}{W_x^d}) - \beta_d)} \tag{5.8}$$

We put $\frac{d}{d(|R_x^a|)} \cdot Y_d(R_x^a) = 0$, and get the critical point,

$$|R_x^a| = \frac{W_x^d}{\delta_d \cdot \tau^{ln(\gamma_d)} \cdot \gamma_d^{\beta_d}}, \text{ where } \tau = \frac{h_d \cdot W_x^d}{\theta_d \cdot \alpha_d}. \tag{5.9}$$

Hence proved. □

Theorem 5.1 *Given benefit function $Y_d(R_x^a)$ for a resource type d on edge server x, the critical point $\frac{W_x^d}{\delta_d \cdot \tau^{ln(\gamma_d)} \cdot \gamma_d^{\beta_d}}$ is a maxima.*

Proof: The second derivative of the benefit function $Y_d(R_x^a)$ is as follows:

$$\frac{d^2}{d(|R_x^a|)^2} \cdot Y_d(R_x^a) = \frac{h_d.W_x^d}{|R_x^a|^2 \cdot ln(\gamma_d)} \left(1 - \left(\frac{\alpha_d}{ln(\gamma_d)} + 1\right) \cdot e^{-(\alpha_d \cdot log_{\gamma_d}(\frac{R_x^a \cdot \delta_d}{W_x^d}) - \beta_d)}\right), \quad (5.10)$$

where $|R_x^a| \geq 1$ as number of assigned users to the edge server, $\alpha_d > 1$ is a tunable parameter, $\beta_d > 0$ is a tunable parameter, $\delta_d > 0$ is average required resource units of type d, $W_x^d > 0$ is the total availabe type d resources of edge server, and $0.90 < \gamma_d < 1$ depends on the average size of offloaded tasks by the users.

As $0 > ln(\gamma_d) < -0.5$ for $0.90 < \gamma_d < 1$, the value of Eq. 5.10 at above discussed parameters will be negative:

$$\frac{d^2}{d(|R_x^a|)^2} \cdot Y_d(R_x^a) = -\text{ive} \quad (5.11)$$

Hence, the critical point $\frac{W_x^d}{\delta_d \cdot \tau^{ln(\gamma_d)} \cdot \gamma_d^{\beta_d}}$ is a maxima for the benefit function of the resource type d at edge server x. □

This maxima point is referred by MAX_x^d for resource type d of edge server x. The maximum value of benefit function for resource type d on edge server x at $MAX_x^d = \frac{W_x^d}{\delta_d \cdot \tau^{ln(\gamma_d)} \cdot \gamma_d^{\beta_d}}$ will be as:

$$Y_d(MAX_x^d) = h_d \cdot W_x^d(ln(\tau) - \beta_d) - \frac{\theta_d \cdot \tau^{\alpha_d}}{e^{\alpha_d(1+\beta_d)}} \quad (5.12)$$

In other words, maxima point MAX_x^d represents the resource type's capacity on an edge server x for given average computational task size, where that edge server delivers the best performance. Thus, using MAX_x^d, the capacity constraint and bottleneck resource for an edge server can be defined as follows.

Definition 5.4 (Bottleneck Resource) *Given an multi-tenant edge server x, the resource type with the capacity to serve the minimum number of users is known as the bottleneck resource. The capacity of the bottleneck resource on edge server x is denoted by MIN_x^{MAX} and determined as follows:*

$$MIN_x^{MAX} = \min_{d \in D}(MAX_x^d). \tag{5.13}$$

The performance of an edge server depends on the bottleneck resource of that edge server. Hence, the capacity constraint of an edge server is defined as follows.

Definition 5.5 (Capacity Constraint) *Given the average computational task size of app users, the capacity constraint for an edge server x is as follows:*

$$|R_x^a| \leq MIN_x^{MAX}, \ \forall x \in S. \tag{5.14}$$

5.3.2 Optimization Model

Given users in $U = \{1, \cdots, n\}$, and servers in $S = \{1, \cdots, m\}$, the objective of this study is to find an optimum server allotment profile $a = (a_1, a_2, \cdots, a_n)$ that maximizes the benefit vector $(Y_1, Y_2, \cdots, Y_m)$. The optimization model for an ERA problem is formally expressed as follows:

$$\max_{a \in A} \sum_{x \in S} Y_x(R_x^a), \ \forall x \in S \tag{5.15}$$

subject to:

$$i \in cov(x), \ \forall i \in U, \forall x \in S \tag{5.16}$$

$$|R_x^a| \leq MIN_x^{MAX}, \ \forall x \in S. \tag{5.17}$$

The objective (5.15) maximizes the resource utilization on the edge servers while reducing the resource overhead. Constraint (5.16) ensures that a user $i \in U$ can be

assigned to a server $x \in S$ only if x covers i. Constraints (5.17) ensures that the number of assigned users to a multi-tenant edge server should not exceed its capacity.

5.4 Distributed Edge Resource Allocation

The previous section determines the critical points in terms of the number of users for each resource type of each edge server where the benefits of utilization are maximum. Using these maxima points, we identify the bottleneck resources on each edge server, defining the capacity constraint. These findings motivate us to design and develop an edge resource allocation algorithm that maximizes edge server utilization while maintaining users' QoS and QoE. This algorithm also aims to balance the load on the edge servers by moving users from overloaded to underutilized edge servers. In our approach, edge servers balance their load by adjusting their geographical coverage area to accommodate different user densities. The variation in the coverage area ranges between the minimum and maximum coverage areas defined by the system. The coverage area of an edge server grows or shrinks depending on its condition, either underutilized or overloaded.

5.4.1 Algorithm

Given U and S, we employ an iterative process for load balancing on edge servers and maximizing their utilization, as described in Algorithm 5.1. Initially, the coverage area of each edge server is set to its minimum (Line1-3). The iterative process starts with the initialization of a random server allotment profile $a = (a_1, a_2, \cdots, a_n)$, where any a_i is an edge server allocated to user i (Line 4). Next, each edge server creates a set R_x^a of users assigned to it (Line 5-7). The current iteration of the process is denoted by t ($t = 1, 2, 3, \cdots$). Term $a(t)$ represents the server allotment profile for the tth iteration. The messages used by the edge server in every iteration are as follows:

- $UNDER\_UTILIZE_x$: An underutilized edge server x broadcasts this message

Algorithm 5.1: App User Allocation Algorithm

Input: $U, \{A_1, A_2 \cdots A_n\}, S$
Output: Server Allotment Profile a

for *each* $x \in S$ **do**
  initialize x's coverage area as $r^x_{cur} =: r^x_{min}$
end
Initialize $a \leftarrow (a_1, a_2, a_3, \cdots a_n)$ to be a random server allotment profile
for *each* $x \in S$ **do**
  $R^a_x \leftarrow \{i \mid i$ is x's user at profile $a\}$
end
repeat
  for *each* $x \in S$ **do**
    if *x is underutilized such that* $|R^a_x| < MIN^{MAX}_x$ *and* $r^x_{cur} + \Delta r < r^x_{max}$ **then**
      edge server x set its coverage range as $r^x_{cur} = r^x_{cur} + \Delta r$
      send a $UNDER\_UTILIZE_x$ message to each user in its coverage area
    end
    if *x is overloaded such that* $|R^a_x| > MIN^{MAX}_x$ **then**
      send a $OVERLOAD_x$ message to each user in its coverage area
    end
  end
  for *each user* $i \in U$ **do**
    if *i gets both* $OVERLOAD_z$ *and* $UNDER\_UTILIZE_y$ *from z and y, respectively* **then**
      if *i's current server allotment decision is such as* $a_i == z$ **then**
        i sends a message ADD^y_i to edge server y
        if $|R^a_y| < MIN^{MAX}_y$ **then**
          y accepts the request of i and updates its user list R^a_y
        end
      end
      if *i's request* ADD^y_i *accepted* **then**
        i sends a message $REMOVE^z_i$ to edge server z
        z accepts i's request and updates its user list R^a_z
        i set its allocation desision $a_i = y$
      end
    end
  end
until *no more server allotment decision updates required*

to all users in its coverage area. It indicates that edge server x is underutilized.

- $OVERLOAD_x$: An overloaded edge server x transmits a message to users in its range. It indicates that edge server x is overloaded.

- ADD_i^x: User i sends a request ADD_i^x to edge server x. This message specifies that user i needs to be allocated to edge server x.
- $REMOVE_i^x$: User i sends a request $REMOVE_i^x$ to edge server x. This message specifies that user i needs to be deallocated from edge server x.

The load balancing process runs in parallel on each edge server in every iteration (Line 9-32). In each iteration, if an edge server x is underutilized, it increases its coverage range by Δr and transmits an $UNDER\_UTILIZE_x$ message to all users who are in its coverage area (Line 10-13). On the contrary, if an edge server x is overloaded, it sends an $OVERLOAD_x$ message to all users in its coverage area (Line 14-16). After transmitting the messages, each user i, who receives both the message and its current allocated edge server is overloaded, participates in the process of updating its server allotment decision (Line 18-32). Suppose a user i receives both messages $UNDER\_UTILIZE_z$ and $OVERLOAD_y$ from edge servers z and y, respectively, where its current server allotment decision is $a_i = z$. To update the server allotment decision, user i first sends a request ADD_i^y to the underutilized edge server y (Line 20-21). In the response, edge server y is allocated to user i, and the user list of y is updated by $R_y^a \cup i$ (Line 22-24). After the allocation of edge server y, user i is deallocated from the edge server z by sending a request $REMOVE_i^z$ to edge server z (Line 26-27). In the response, edge server z removes the user i from its user list and updates R_z^a by $R_z^a \setminus i$ (Line 28). User i then sets its server allotment decision a_i by y (Line 29). This iterative process will terminate if no user updates his server allotment decision. Eventually, this process returns the coverage area of each edge server to accommodate different user densities, and an server allotment profile specifies the edge server allocation to users.

5.5 Numerical Evaluation

This section assesses our proposed algorithm through executing numerous experiments on various combinations of users, users resource requirements, and edge servers' com-

puting capacities.

5.5.1 Performance Benchmark

Our proposed approach is compared to five state-of-the-art approaches given below to solve the same problem discussed in this chapter.

- A-PBRA [92]:It assigns edge computing resources based on the end users' urgencies and priorities. It aims to maximize the utilization of edge computing resources and incoming user requests.
- DoSRA [91]: This method allocates the edge resources in a decentralized manner. DoSRA aims to improve the Quality of Service (QoS) and maximize resource utilization.
- PBRA [2]: This approach allocates the edge resources to the users of different services using the distributed game-theoretic technique. Its objective is to assign the resources cost-effectively.
- GT-EUA [41]: Using the distributed game-theoretic technique, it assigns edge resources to users while optimizing the allocated users to edge servers and system cost.
- MCFH [42]: This research study proposed a heuristic approach to assign users to servers. It maximizes the number of allocated users to servers while minimizing their cost.

Python 3.5 is used to write all of the experiments. These experiments were carried out on a Windows 10 machine with an Intel CORE i7-7700U CPU running at 3.60 GHz and 8 GB of RAM.

5.5.2 Simulation Settings

Generally, the research work in this field uses a custom-built simulator in various languages for different experiments. We created our simulation setting in Python 3.5 to

Table 5.1: Experimental Settings

	Number of users	Resource requirement per user (δ_d)	Number of servers	Available Resources per server (μ_d)
set-1	1000,...,10000	1, 1.5, 2	100	25
set-2	1000,...,10000	1.5	100	25
set-3	5000	1.5	100	5, $\cdots$ 50

carry out experiments in the same way earlier chapters [41,42] in this research field did. We create a 21 km x 21 km region to test the method and randomly place 100 edge servers throughout this space. The available resources of each resource type of every server are randomly generated by a normal distribution $N(\mu_d, \sigma_d^2)$, where and $\sigma_d = 5$ denotes the standard deviation, and $\mu_d = 25$ represents the average resource units. Any resource type can generate a negative amount under the normal distribution; this is rounded to 1. D = {Cache, Memory, Storage, CPU} is a resource type set in our experiments. The user-offloaded task's size is determined by a uniform distribution over [10 KB, 60 KB]. The utilization ($Z_d(x)$) of every resource type d of server x is determined by parameter γ_d and user list R_x^a. The average task size determines the value of γ_d. In our experiment, for each resource type d, the value of γ_d varies from 0.90 to 0.99. We use Round Robin scheduling with an 8 ms time quantum to allocate the CPU to offloaded tasks. Tables 5.1 and 5.2 summarize the experimental and parameter settings. The results of each experiment are averaged after 50 repetitions to neutralize extreme cases such as dense or sparse user/edge server distributions.

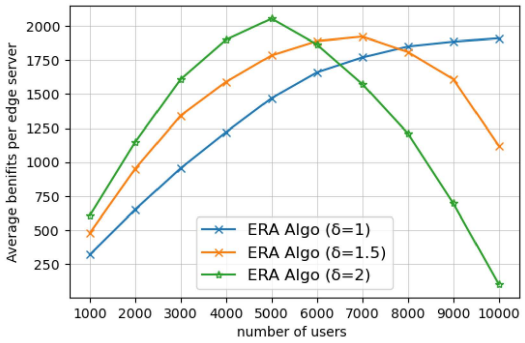
5.5.3 Performance Comparison

We assess the ERA algorithm's performance using three different experimental settings listed in Table 5.1. The performance of the proposed algorithm is compared with others in each experiment using the following metrics.

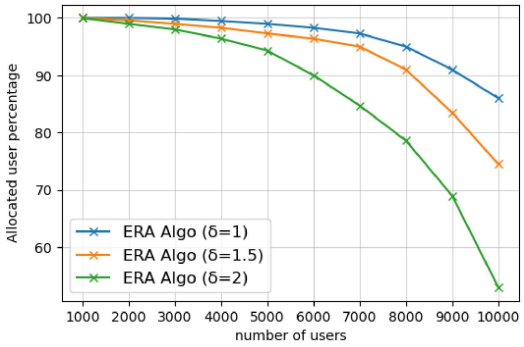
- *Benefits:* The benefits of an edge server represent the difference between generated revenue and the Overhead cost.

Table 5.2: Parameters' Setting

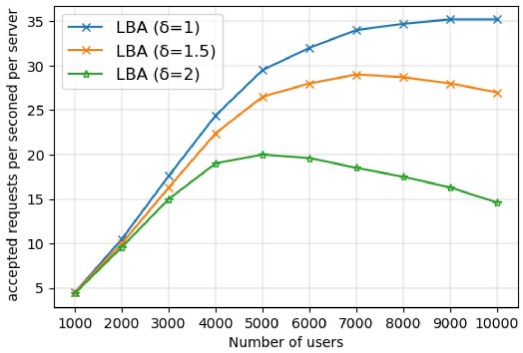
Name	**Description**
Resource Type d	$d \in D = \{$Storage, Cache, Memory, CPU$\}$
Offloaded task size	uniform distribution over $[10\ \ KB,\ \ 60\ \ KB]$
Time interval between generated tasks by a user	uniform distribution over $[1\ \ second,\ \ 5\ \ second]$
Resource utilization tunable parameter γ_d	$\gamma_d \in \{0.91, 0.92, 0.93, 0.94, 0.95\}$
Assigned preference weight h_d to each resource type d	0.25
process transition time on CPU	2 ms
Assigned QoS weight θ_d to each resource type d	0.25
Tunable parameter α_d	0.4
Tunable parameter β_d	2
CPU Scheduling Algorithm	Round Robin Algorithm
Time quantum	10 ms
Page replacement algorithm	LRU
Memory size	256 KB
Page size	8 KB
memory access time	1 ms
Minimum coverage area r^x_{min}	750 meters
Maximum coverage area r^x_{max}	1000 meters



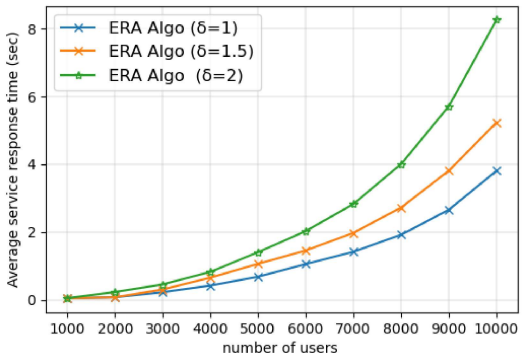
(a) Users vs. server's benefits.



(b) Users vs. allocated users.



(c) Users vs. accepted requests.



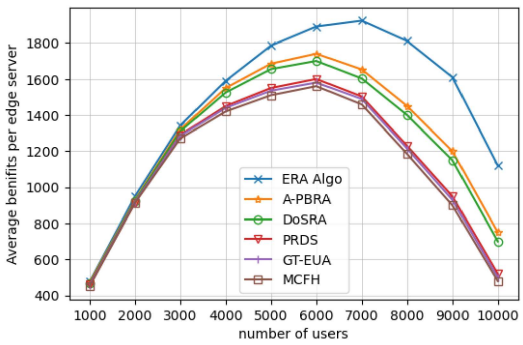
(d) Users vs. response time.

Figure 5.4: Experiment 1 corresponds to Set#1 and shows the ERA algorithm's performance.

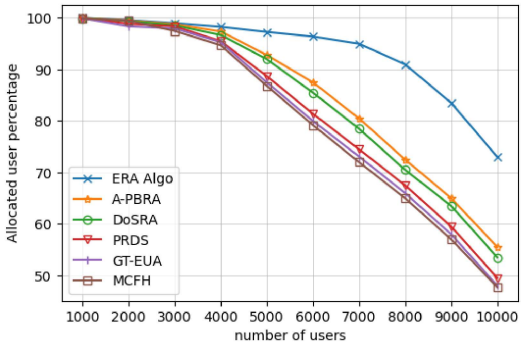
- *Allocated users:* The percentage of total users assigned to edge servers is represented by this metric.
- *Accepted request:* The average number of task offloading request served by each edge server in every second.
- *Response time:* This represents the typical delay between submitting a task and receiving the first response from an edge server. Response time is measured in our tests as the time spent in a waiting queue for initial CPU allocation.

5.5.3.1 Experiment 1

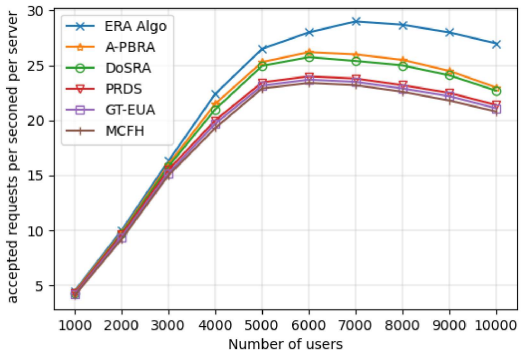
The first experiment compares the performance of the ERA algorithm for different average user resource requirements versus user count, as shown in Fig 5.4. Set-1 of



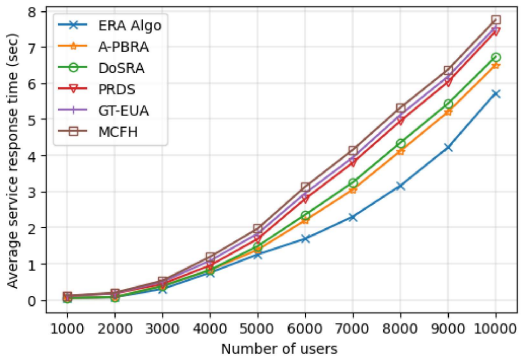
(a) Users vs. server's benefits.



(b) Users vs. allocated users.



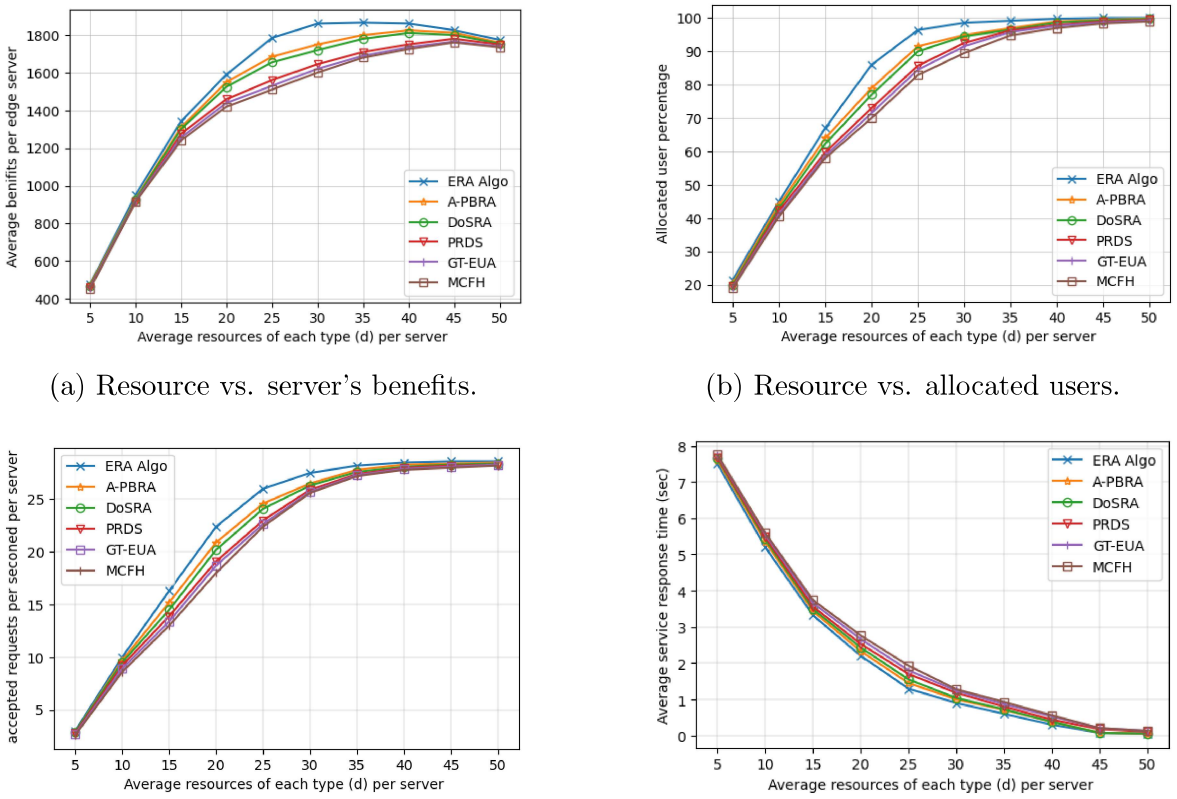
(c) Users vs. accepted requests.



(d) Users vs. response time.

Figure 5.5: Experiment 3 show the ERA algorithm's performance compared with other approaches. This experiment corresponds to Set#2.

Table 5.1 contains the settings for this experiment. The findings depicted in Fig. 5.4a show that the average benefit of an edge server rapidly rises and falls with the number of users in the case of higher resource-demanding users compared to lower resource-demanding users. It is because users with high resource demand offload the heavy computational tasks that cause the edge servers to become quickly overloaded, and the edge server performance begins to degrade after a certain point, e.g., 5×10^3 users for $\delta = 2$ in our experiments. It is apparent from Fig. 5.4b that more users with low resource demands ($\delta = 1$) can be accommodated on the same edge servers than users with higher resource demands ($\delta = 2$) as users with ($\delta = 1$) have light computational tasks to offload. As a result, after a certain point, e.g., 4×10^3 in our experiment, the edge servers can handle more requests from lower-demanding users than higher-



(a) Resource vs. server's benefits.

(b) Resource vs. allocated users.

(c) Resource vs. accepted requests.

(d) Resource vs. response time.

Figure 5.6: Experiment 3 show the ERA algorithm's performance compared with other approaches. This experiment corresponds to Set#3.

demanding users, as shown in Fig. 5.4c. The results illustrated in Fig. 5.4d show that the response time is longer for highly resource-demanding users because their time spent in the waiting queue is longer due to resource unavailability.

5.5.3.2 Experiments 2

Set-2 in Table 5.1 contains the settings regarding the second experiment, in which the ERA algorithm's performance is compared to state-of-the-art approaches with increasing number of users, as shown in Fig 5.5. Before explaining the results, we note that when the number of users reaches a certain threshold with the available edge servers, an uneven distribution of users may result in some edge servers being underutilized while others are overloaded. The average benefits per edge server are shown in Fig. 5.5a

as the number of users increases. The ERA outperformed A-PBRA, DoSRA, PRDA, GT-EUA, and MCFH at some point, e.g., 4000 in our experiments. This is because the ERA moves users from overloaded edge servers to underutilized edge servers. Fig. 5.5b shows the percentage of the allocated users against the number of users. The ERA performs better compared to state-of-the-art approaches. This happens as the ERA allows underutilized edge servers to serve more users by expanding the coverage area of edge servers. As illustrated in Fig. 5.5c, in the case of ERA, edge servers handle offloaded tasks well because they execute the more offloaded tasks by moving the users from overloaded to underutilized edge servers. The results presented in Fig. 5.5d illustrate that service response time increases with an increasing number of users. In the case of ERA, edge servers respond quickly to users because each edge server performs more offloaded tasks per unit time. As a result, the amount of time spent in the resource waiting queue is reduced.

5.5.3.3 Experiment 3

In the third experiment, the performance of the ERA algorithm is compared to the state-of-the-art approaches with the increase of edge servers' computing resources, as shown in Fig. 5.6. Set-3 in Table 5.1 lists the settings regarding this experiment. The results shown in Fig. 5.6a depict the average benefits of the edge servers. The average benefits of edge servers are roughly the same in all approaches up to a point, e.g., 15 in our experiment, because almost all edge servers are overloaded due to fewer available resources. After this point, as computing resources increase, some edge servers may become underutilized in areas with sparse user distribution. In this case, ERA algorithm outperforms others (between 15 to 45 in our experiment) because it allows an underutilized edge server to provide services to users of overloaded edge servers by expanding their coverage area. After a certain point, e.g., from 45 in our experiment, almost no edge server is overloaded. As a result, the benefits of edge servers after this

point are the same across all approaches. Furthermore, edge server benefits begin to decline at some point in all approaches, for example, at 40 in our experiment, because some resources on the edge servers may go unused due to an abundance of available resources. Fig. 5.6b shows that the ERA performs better compared to state-of-the-art approaches in the case of user allocation. This happens as the ERA allows underutilized edge servers to serve more users by expanding the coverage area of edge servers. As shown in Fig. 5.6c, the number of offloaded jobs handled by edge servers in the ERA is more prominent than in other cutting-edge techniques since it balances the load on the edge servers. As the edge servers handle more offloaded tasks in the case of ERA, the edge servers quickly respond to users' offloaded tasks, as shown in Fig. 5.6d.

5.6 Summary

In this chapter, we propose an approach for allocating edge resources in a distributed manner that deals with the geographically dynamic user density and bottleneck resources on the edge server. Our proposed approach formulates the Edge Resource Allocation (ERA) problem as the benefit function of edge servers' resources, which is defined by the revenue generated by the resource utilization and the cost of degrading the Quality of Service (QoS). We then proposed a ERA algorithm that accommodates the IoT and mobile users from overloaded to underutilized edge servers. This way, ERA algorithm improves the QoS by reducing the latency experienced by the users and the Quality of Experience (QoE) by maximizing the number of served users. Experiment results show that ERA algorithm archives minimum latency while serving maximum users compared to state-of-the-art approaches, which is a significant advancement.