Chapter 4

Resource Allocation to App Vendors

The previous chapter addresses the problem of allocating app users to resources hired by the app vendor on various multi-tenant edge servers. This establishes a trade-off between the cost and the QoS. In this chapter, we investigate the problem of allocating edge computing resources to various app vendors competing for the same resources. This chapter aims to lower the cost while maximizing the resource utilization of multi-tenant edge servers.

4.1 Introduction

App vendors must hire the resources on edge servers in a manner that maximizes the app users assigned to edge servers in order to maximizes benefits. App vendor also pays for the computing resources hired on the edge servers [129, 130]. Thus, the other objective of each app vendor is to hire the optimal resources of the edge servers to reduce the cost. This can be accomplished by maximizing resource utilization of multi-tenant edge servers and minimizing the required edge servers. Multi-tenant architecture, commonly referred to as multi-tenancy, is a software architecture in which app instances of different

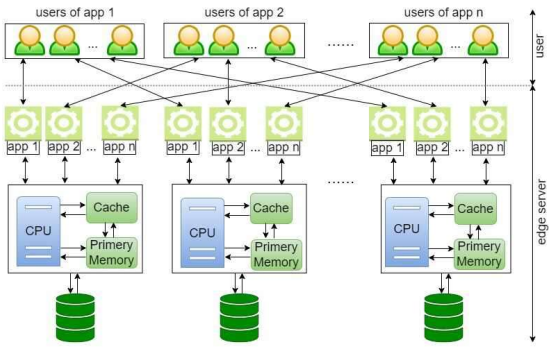


Figure 4.1: Multi-tenant architecture of edge servers.

applications run on a single physical machine where every single app instance serves multiple users, as shown in Fig. 4.1. We refer to this problem of allocating resources to different app vendors (services) as the Edge Resource Allocation (ERA) problem. The ERA problem is an NP-hard problem [26, 41].

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

The proposed approach maximize resource utilization by effectively leveraging the multi-tenant edge servers, reducing the required edge servers for each app vendor. As a result, it minimizes the cost incurred by each app vendor. Furthermore, this solution increases the overall assigned app users to given resources as it maximizes resource utilization. The key to maximizing resource utilization of any edge server is increasing the number of users using the same app on the edge server. With more users of the same service on each multi-tenant edge server, edge servers can be more effectively

utilized since users can share global data variables, program code, cache contents, and other resources on a wide scale. As a result, this resource allocation also reduces service response time and latency.

The main contributions of this chapter are as follows:

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

The rest of the chapter is organized as follows. Section 4.2 elaborates on the ERA problem along with the system model. Section 4.3 formulates the ERA problem as an ERAGame and analyzes the game property. Section 4.4 presents the ERA algorithm to achieve the PNE. Sections 4.5 and 4.6 analyze the ERA algorithm's performance.

4.2 System Model

To solve the ERA problem from the perspective of app vendors, we consider n app vendors $V = \{1, 2, \cdots, n\}$, and each app vendor i has i_ρ app users $U_i = \{u_{i_1}, u_{i_2}, u_{i_3}, \cdots u_{i_\rho}\}$ at different locations. The app vendors' job is to hire computing capacity from m edge servers $S = \{1, 2, \cdots, m\}$ at different locations to serve app users. Various edge infrastructure providers own these edge servers. This chapter only studies resource utilization from the perspective of the computing capacities belonging to the edge servers. The total available computing capacity of an edge server x is denoted by $W_x = (W_x^d)$, where $d \in D = \{$ cache, memory, CPU, storage, $\cdots\}$. Fig. 4.1 depicts the organization of these various computing capacities. We assume that all the computing resources are managed at the edge server's operating system level. Our proposed approach works for any employed application, and it is independent of how the operating system manages computing capacities.

For numerical evaluation of the proposed approach, we create the simulation environment by considering some assumptions [131, 132] as follows. 1) The app vendors deploy the computational intensive calculation-based apps on the edge servers. 2) The tasks offloaded to an edge server by app users are queued to be executed on the CPU using Round Robin scheduling. 3) Before beginning the execution of an offloaded task, the task's availability in the cache memory is checked. If any component of the task is not already in cache memory, it is loaded into the cache memory using Least Recently Used algorithm.

Similarly, our proposed approach can be applied in other settings, and the results will be consistent. We examine the ERA problem in quasi-static scenarios where app users do not change computing capacity requirements and locations while allocating edge servers' resources, similar to other chapters [41, 42, 133].

Definition 4.1 (Allocation Decision) *Given an app vendor i, $U_i = \{u_{i_1}, \cdots u_{i_\rho}\}$,*

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

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

$$u_{i_k} \in cover(x), \quad \forall u_{i_k} \in U_i, \forall i \in V, \forall x \in S \tag{4.1}$$

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

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

Definition 4.2 (Allocation Decision Profile) *Given* $V = \{1, \cdots, n\}$*,* $U = \{U_1, \cdots U_n\}$*, and* $S = \{1, \cdots, m\}$*, an allocation decision profile is the* $n \times m$ *matrix (denoted by* a*), in which the element at* i*th row and* x*th column is* $\beta_{i,x}$*. In other words, an allocation decision profile* $a = (a_1, a_2, a_3 \cdots a_n)$ *is the collection of all app vendors allocation decisions, where* $a \in A = A_1 \times A_2 \times \cdots \times A_n$*.*

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

4.2.1 App Vendor Cost Model

The infrastructure providers rent out the computational resources based on a pricing model. The cost incurred by an app vendor to hire the computing resources will com-

prise two components: usage cost and fixed cost. In this chapter, we use a cost model that determines the usage cost of computing resources based on their utilization. The pricing model based on resource utilization is called the pay-as-you-go pricing model used by different infrastructure providers, e.g., AWS, Azure, Salesforce, etc. The utilization of resources of an edge server hired by an app vendor can be determined from Eq. 3.5 in Chapter 3. If an app vendor i hires computing resources $\beta_{i,x} = (\beta^d_{i,x})$, $d \in D$ of edge server x, the usage cost, denoted by $B(\cdot)$, can be determined as follows:

$$B(\beta_{i,x}) = \sum_{d \in D} h_d \cdot Z(\beta^d_{i,x}) \cdot \beta^d_{i,x} \tag{4.2}$$

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

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

$$F(a, x) = \frac{f_x}{|R^a_x|}, \tag{4.3}$$

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

the overall utilization of an edge server subject to capacity constraints.

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

$$c_{i,x}(a) = \frac{f_x}{|R_x^a|} + B(\beta_{i,x}). \tag{4.4}$$

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

$$c_i(a) = \sum_{x \in \nu_i(a)} c_{i,x}(a) = \sum_{x \in \nu_i(a)} \left(\frac{f_x}{|R_x^a|} + B(\beta_{i,x}) \right) \tag{4.5}$$

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

$$C(a) = \sum_{i \in V} c_i(a) \tag{4.6}$$

4.2.2 Optimization Model

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

$$\min_{a \in A} \sum_{i \in V} c_i(a) \tag{4.7}$$

subject to:

$$u_{i_k} \in cover(x), \quad \forall u_{i_k} \in U_i, \forall i \in V, \forall x \in S \tag{4.8}$$

$$\sum_{i=1}^{n} \beta_{i,x}^{d} \leq W_{x}^{d}, \forall x \in S, \forall d \in D \tag{4.9}$$

$$\beta_{i,x}^{d} \geq 0, \forall d \in D, \forall i \in V, \forall x \in S \tag{4.10}$$

Objective 4.7 minimizes the app vendors' total cost. Constraint 4.8 ensures that every app user u_{i_k} should be allocated to an edge server x only when x covers u_{i_k}. Constraints 4.9 and 4.10 ensures that the computing capacity hired by app vendors on an edge server x must not exceed of available computing capacities of x.

4.3 Edge Resource Allocation Game

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

Definition 4.3 (Pure Nash Equilibrium) *An allocation decision profile* $a^* = (a_1^*, a_2^*, a_3^* \cdots a_n^*)$ *is a PNE if no app vendor can reduce its cost by unilaterally deviating from its allocation decision, i.e,*

$$c_i(a_i^*, a_{-i}^*) \leq c_i(a_i, a_{-i}^*) \text{ for all } a_i \in A_i, i \in V \tag{4.11}$$

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

$$\min_{a_i \in A_i} c_i(a_i, a_{-i}) \tag{4.12}$$

ERAGame formulates the problem as a tuple $(V, \{A_i\}_{i \in V}, \{c_i\}_{i \in V})$, where V is a set of app vendors, A_i is i's finite set of allocation decisions subject to Eqs. 4.8-4.10, and c_i is the cost function that determine the cost of i's selection decision $a_i \in A_i$. In ERAGame, all the app vendors gradually move towards a PNE to reducing their costs. Whether ERAGame accepts at least one PNE is of critical significance to this study.

4.4 Distributed Resource Allocation Mechanism

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

To achieve fast convergence, we first partition the edge server set into different groups and feed these groups to the ERA algorithm as an input. The edge server groups must be created in such a way that resource allocation decisions made on one

edge server in a group do not directly impact decisions made on other edge servers within that group. This kind of grouping will allow the ERA algorithm to run parallel on all the edge servers within each group, leading to fast convergence to PNE. However, the existence and quality of PNE in ERAGame are irrespective of the process of grouping and the number of edge server groups. The edge servers are partitioned into the groups as described above in Section 3.5.1 of Chapter 3. Hence, the edge server set S is partitioned into groups as $T = \{T_1, T_2, \cdots, T_M\}$, where any T_l denotes the lth 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 $(x_{k_1}, x_{k_2}) \in T_l$, for every $T_l \in T$
- $T_{l1} \cap T_{l2} = \emptyset$, for every $T_{l1}, T_{l2} \in T$

4.4.1 Edge Resource Allocation Algorithm

Given $V = \{1, \cdots, n\}$, $\{U_i, \cdots, U_n\}$ and $S = \{1, \cdots, m\}$, ERAGame applies an iterative process for convergence to the PNE, as shown in Algorithm 4.1. In each iteration, app vendors try to minimize their costs by following the ERAGame rules while gradually moving towards the PNE. The process begins with an arbitrary allocation decision profile a, and an app vendor list R_x^a is created for every edge server (Lines 1-2). The process iterations are repeated for every group until the system reaches PNE. Each iteration is represented by $t(t = 1, 2, 3 \cdots)$, and the allocation decision profile in any t is denoted by $a(t)$.

In every iteration, each edge server sends a message containing $< SI_x, R_x^{a(t)}, \lambda_x^{a(t)} >$ to neighboring edge servers (Line 4), where SI_x is the identity of edge server x, $R_x^{a(t)}$ is the list of app vendors, and λ_x^a is the available resources of edge server x at allocation decision profile $a(t)$. This message is transmitted in a decentralized manner which needs messaging synchronization [124]. Next, a group T_l is selected, where $l \equiv t \pmod{T}$ (Line 5). The computation in each iteration of the allocation process (Lines 6-20) is performed by individual app vendors on each edge server $y \in T_l$ in parallel.

Algorithm 4.1: Edge Resource Allocation

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

1 **Initialize** $a \leftarrow (a_1, a_2, a_3, \cdots a_n)$ to be an arbitrary allocation decision profile
2 At every edge server $x \in S$ do
$R_x^a \leftarrow \{i \mid i$ is an app vendor hired resources on x at $a\}$
3 **repeat**
4 each edge server $x \in S$ send a message containing $< SI_x, R_x^{a(t)}, \lambda_x^{a(t)} >$ to its neighbouring edge servers
5 selection of edge server group T_l for iteration t
6 **for** *each edge server* $x \in T_l$ **do**
7 **for** *each app vendor* $i \in R_x^{a(t)}$ **do**
8 i finds each set of alternative edge servers NEG_x of x, subject to constraints 4.8, 4.9, and 4.10
9 **for** *each set of edge servers* NEG_x **do**
10 i calculates the possible cost of the resource allocation on each edge server in NEG_x as in Eq. 4.4
11 i finds an edge server set NEG_x^{min} such that $\min_{NEG_x}(\sum_{y \in NEG_x} c_{i,y}(a(t)))$
12 **if** $\sum_{y \in NEG_x^{min}} c_{i,y}(a(t)) < c_{i,x}(a(t))$ **then**
13 app vendor i request for resource allocation on each $y \in NEG_x^{min}$
14 **for** *each* $y \in NEG_x^{min}$ **do**
15 **if** *y receive request from multiple vendors* **then**
16 edge server y accepts the requests that best fit the available resources
17 **else**
18 accept the request of app vendor i
19 **if** *app vendor i's request accepted* **then**
20 deallocate the resources from x
21 **until** *no more allocation decision updates needed*;

After selecting group T_l, each app vendor i with resources on edge servers in group T_l can participate in the current iteration to minimize its cost (Line 6-7). Then each participant app vendor i on each edge server $x \in T_l$ finds the set of optimal alternative edge servers NEG_x of x in terms of cost. This optimal set must satisfy the proximity constraints Eq. 4.8 and capacity constraints Eqs. 4.9-4.10. This means that the edge servers in NEG_x can give the services to i's app users currently getting the services

from x (Line 8). Next, if the app vendor i's possible cost on edge servers in NEG_x is less than the cost incurred on x, the app vendor i requests resource allocation on the edge servers in NEG_x (Lines 9-13). The resources are allocated to any app vendor on every edge server $y \in NEG_x$ according to (Lines 14-18). If any edge server receives multiple resource allocation requests from app vendors, it accepts the requests that can best fit the available resources $\lambda_x^{a(t)}$. In this case, an edge server y can accept the request of app vendor i if it is included in the best fit. If app vendor i's request is accepted by all edge servers in $y \in NEG_x$, the allocated resources on edge server x for the services provided by app vendor i are deallocated (Lines 19-20). These iterations repeat for each group $T_l \in T$ until the app vendors stop updating their selection decisions (Line 21). App vendors' individual selection decisions comprise the final allocation decision profile that will be a solution as the PNE.

4.5 Theoretical Analysis

4.5.1 Game Property

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

using the ERA algorithm. The key is to find a potential function and to establish that ERAGame is a potential game defined as follows:

Definition 4.4 (Potential Game) *A game is a potential game if there is a potential function* $\Phi(a)$ *such that,*

$$c_i(a_i', a_{-i}) \leq c_i(a_i, a_{-i}) \Rightarrow \Phi(a_i', a_{-i}) \leq \Phi(a_i, a_{-i}) \tag{4.13}$$
$$\text{for any } i \in V, a_i, a_i' \in A_i \text{ and } a_{-i} \in \prod_{l \neq i} A_l$$

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

$$\Phi_x(a) = \sum_{k=1}^{|R_x^a|} (\frac{f_x}{k}) + \sum_{i \in R_x^a} B(\beta_{i,x}) \tag{4.14}$$

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

$$\begin{aligned} \Phi(a) &= \sum_{x \in S} \left(\sum_{k=1}^{|R_x^a|} (\frac{f_x}{k}) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \\ &= \sum_{x \in S} \left(f_x \cdot H(|R_x^a|) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \end{aligned} \tag{4.15}$$

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

Theorem 4.1 *The ERAGame with the potential function $\Phi(a)$ is a potential game.*

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

$$\Delta c_i = c_i(a') - c_i(a) < 0 \tag{4.16}$$

Due to deviation, the list of edge servers on which app vendor i has computational resources is updated from $\nu_i(a)$ to $\nu_i(a')$, where $\nu_i(a') \setminus \nu_i(a)$ and $\nu_i(a) \setminus \nu_i(a')$ contain all the edge servers added and left by i, respectively. Only edge servers in $\nu_i(a') \setminus \nu_i(a)$ and $\nu_i(a) \setminus \nu_i(a')$ impact the cost of app vendor i. The change in app vendor i's cost from Eqs. 4.5 and 4.16 can be elaborated as,

$$\begin{aligned}\Delta c_i &= \sum_{x \in \nu_i(a') \setminus \nu_i(a)} c_{i,x}(a) - \sum_{y \in \nu_i(a) \setminus \nu_i(a')} c_{i,y}(a') \\ &= \sum_{x \in \nu_i(a') \setminus \nu_i(a)} \left(\frac{f_x}{|R^a_x| + 1} + B(\beta_{i,x}) \right) \\ &\quad - \sum_{y \in \nu_i(a) \setminus \nu_i(a')} \left(\frac{f_y}{|R^a_y|} + B(\beta_{i,y}) \right) \; < 0 \end{aligned} \tag{4.17}$$

Where R^a_x or R^a_y is a list of app vendors that hire computing resources of x or y, respectively, before app vendor i deviates. The change in Potential function value is,

$$\Delta\Phi \;\; = \;\; \Phi_{Inc} - \Phi_{Dec} \tag{4.18}$$

The app vendor i hires the computing resources on the edge servers in $\nu_i(a') \setminus \nu_i(a)$. As a result, the potential of these edge servers will increase. Therefore, from Eqs. 4.14 and 4.15, the increase in potential is $\Phi_{Inc} = \sum_{x \in \nu_i(a') \setminus \nu_i(a)} \left(\frac{f_x}{|R^a_x|+1} + B(\beta_{i,x}) \right)$. On the other hand, i abandons the computing resources of edge servers in $\nu_i(a) \setminus \nu_i(a')$. Hence, from Eqs. 4.14 and 4.15, the value of Φ decreases by $\Phi_{Dec} = \sum_{y \in \nu_i(a) \setminus \nu_i(a')} \left(\frac{f_y}{|R^a_y|} + B(\beta_{i,y}) \right)$.

Thus, from Eqs. 4.17 and 4.18,

$$\Delta\Phi = \Delta c_i < 0 \tag{4.19}$$

From Eq. 4.19, if any app vendor i changes its allocation decision to minimize its cost, $c_i(a') \leq c_i(a)$ implies $\Phi(a') \leq \Phi(a)$. Hence, ERAGame is a potential game. □

Theorem 4.1 also proves that the potential function's value decreases with each change in the allocation decision made by any app vendor as an improvement. In the ERA algorithm, app vendors make the allocation decision to minimize their costs. Therefore, the value of the potential function will decrease with every iteration of the ERA algorithm.

Theorem 4.2 *In ERAGame, the system converges to at least one PNE.*

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

1. The ERAGame contains a finite number of allocation decision profiles, and the potential function $\Phi(a)$ is defined on them. Consequently, the outcomes of $\Phi(a)$ are also finite.
2. According to Theorem 4.1, the value of the potential function $\Phi(a)$ declines monotonically with each change in the app vendors' allocation decisions under the ERA algorithm; thus, no cycle is feasible in the ERAGame under the ERA algorithm.

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

4.5.2 Convergence Analysis

This subsection analyzes how many iterations of the ERA algorithm are required to converge at PNE by following the trajectory of the convergence process. The potential function is used to observe the convergence process of the ERA algorithm, as aforementioned. In each iteration of the ERA algorithm, the app vendors make the allocation decision for their cost reduction and collectively generate an allocation decision profile. This process continues until the system reaches the PNE. The app vendors reduce their costs in every iteration of the ERA algorithm, so the potential $\Phi(a)$ value would monotonically decrease in every iteration, as demonstrated in Theorem 4.1. This implies that the allocation decision profile is not repeated during the execution of the ERA algorithm employed in the finite ERAGame. Theorem 4.2 proved that the system reaches the PNE in a finite number of iterations of the ERA algorithm. Moreover, the edge server set is partitioned into T groups. In each iteration of the ERA algorithm, app vendors can make allocation decisions in parallel on every edge server in any group $T_l \subseteq T$. From these conclusions, it follows that the number of iterations in the ERA algorithm to reach Nash equilibrium will never exceed $\max_{q_{i,x}}(T \times q_{i,x})$, where $q_{i,x}$ is the number of all possible alternates in place of an edge server x for app vendor i.

4.5.3 Price of Stability

This subsection evaluates the bound on the quality of the PNE solution obtained using the ERA algorithm in order to meet app vendors' optimization objectives, as discussed in Section 3.3. The best PNE has the lowest overall cost of app vendors. In general, the Price of Stability (PoS), which is the ratio of the overall cost of app vendors at best PNE to the centralized optimum, demonstrates the quality of the PNE. If allocation decision profiles a^* and a^o are the best PNE solution and centralized solution, respectively, the PoS is,

$$PoS(a) = \frac{C(a^*)}{C(a^o)} \tag{4.20}$$

Where $C(a^*)$ and $C(a^o)$ are overall costs incurred by app vendors at allocation decision profiles a^* and a^o, respectively. Since the potential function plays a vital role in the estimation of PoS, we first establish a relationship between the potential function and the total cost of app vendors.

Lemma 4.1 *In ERAGame, for any allocation decision profile,* $C(a) \leq \Phi(a) \leq H(n) \cdot C(a)$*, where* $C(a)$ *is the total cost incurred by* n *app vendors.*

Proof: From Eq. 4.7, n app vendors' total cost $C(a)$ is,

$$C(a) = \sum_{x \in S} \left(f_x + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) \tag{4.21}$$

As defined in section 4.5.1, $H(|R_x^a|) \geq 1$ for any $|R_x^a| \geq 1$. Thus,

$$C(a) \leq \sum_{x \in S} \left(f_x \cdot H(|R_x^a|) + \sum_{i \in R_x^a} B(\beta_{i,x}) \right) = \Phi(a) \tag{4.22}$$

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

$$\Phi(a) \leq H(n) \cdot \sum_{x \in S_a} (f_x + \sum_{i \in R_x^a} B(\beta_{i,x})) = H(n) \cdot C(a) \tag{4.23}$$

Hence, from Eqs. 4.22 and 4.23, $C(a) \leq \Phi(a) \leq H(n) \cdot C(a)$. □

Theorem 4.3 *In ERAGame, the price of stability is at most H(n).*

Proof: Suppose the allocation decision profile a^o is a central enforced optimum solution and $C(a^o)$ is the cost at a^o. We assume that app vendors begin with a^o as the initial

allocation decision profile and changing allocation decisions until the system reaches a PNE a^*. As stated in the Theorem 4.1, $\Phi(a)$ will decrease with every improvement of app vendors in this process, so,

$$\Phi(a^*) \leq \Phi(a^o) \tag{4.24}$$

From the first inequality of lemma 4.1,

$$C(a^*) \leq \Phi(a^*) \tag{4.25}$$

From inequalities in Eqs. 4.24 and 4.25,

$$C(a^*) \leq \Phi(a^o) \tag{4.26}$$

The second inequality of lemma 4.1 says,

$$\Phi(a^o) \leq H(n) \cdot C(a^o) \tag{4.27}$$

Hence, the result infers from the inequalities in Eq. 4.26 and 4.27,
$C(a^*) \leq H(n) \cdot C(a^o) \Rightarrow \frac{C(a^*)}{C(a^o)} \leq H(n)$ □

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

4.6 Numerical Evaluation

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

4.6.1 Performance Benchmark

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

- Best-Fit: This approach allocates the edge server resources to an app vendor in such a way that the remaining resources are as small as possible, subject to proximity constraints.
- Greedy: This approach allocates the resources to any app vendor on an edge server with the maximum available resources, subject to proximity constraints.
- PRDS [2]: The resource allocation problem for various services with budget limitations was solved using a Game-Theoretic approach. Its goal is to allocate the optimal bundle of resources to each service.
- TPDS [41]: This research proposed a game-theoretic approach for allocating app users to edge servers. Its goal is to increase the number of app users assigned to the edge servers while lowering the overall cost.
- MCFH [42]: his solution uses a heuristic to solve the app user allocation problem by first allocating the highest capacity edge server. Its objective is to maximize the number of app users on edge servers while reducing the number of edge servers required.

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

Table 4.1: Experimental Settings

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

4.6.2 Simulation Settings

In this area, the works generally use the custom-built simulator in different languages. In the same way, as other chapters [19-20] in this area used to develop the simulation environment, we created our environment in Python 3.5 to implement all of the experiments. We have set up a 20 km x 20 km rectangular space to evaluate the algorithm. The base stations are spaced 600 meters apart throughout this rectangular region. For each simulation, we deploy 100 edge servers at different base stations at random. Edge servers' coverage radius is uniformly distributed over [750 m, 1500 m]. The available resources on each edge server are produced at random using a normal distribution $N(\mu, \sigma^2)$, with $\mu = 50$ being the average amount of each resource type d, and $\sigma = 10$ representing the standard deviation. Each type's generated number of computing resource units is rounded off to a positive integer number. Under the normal distribution, a negative amount of any resource type can be generated; hence any negative value is rounded to 1. In our experiments, we consider resource type set $D = \{bandwidth,\ Storage,\ Memory,\ Cache,\ CPU\}$. The app users' offloaded work is generated by uniform distribution over [10KB, 60KB]. The number of app users R_x^a and parameter γ_d determine the resource consumption $Z_d(x)$ of type d on server x. The average task size, denoted by Avg_Size, offloaded to edge server determines the value of γ_d. For each resource type d in our experiment, the value of γ_d is set to $\{(10KB \leq Avg\_Size < 20KB, \gamma_d = 0.91),\ (21KB \leq Avg\_Size < 30KB, \gamma_d = 0.92),\ (31KB \leq Avg\_Size < 40KB, \gamma_d = 0.93),\ (41KB \leq Avg\_Size < 50KB, \gamma_d = 0.94),\ (50KB \leq Avg\_Size < 60KB, \gamma_d = 0.95)\}$. Priority I_d is set to 0.25 for each resource

type d. Any app user's resource requirements are determined by the size of the offloaded task. We presume that a task with less than 30 KB requires one unit of each resource type, whereas a task with a size of more than 30 KB requires two. The fixed cost of each edge server is generated by uniform distribution over [50, 75]. We create the simulation setting to calculate the response time as in the chapter [131, 132]. We assume a cache memory of 128 kb in size, with each block being 4kb in size. Each app user's computational task is divided into pages of 4 kb in size. We assume that for each app, the first 50% of total pages of computational tasks offloaded by any app user is shared with other app users of the same app. The Least Recently Used (LRU) page replacement technique swaps the pages from the cache. The Round Robin scheduling algorithm is used for CPU allocation with a time quantum of 10 ms. The response time for an app user is calculated as the amount of waiting time in the queue for first-time CPU allocation and the time required to cache all of its computational task's pages. We assume that every 1 kb of the offloaded task by app users needs a CPU burst of 1 ms. The cache access (page hit) and primary memory access time (page miss) are assumed to be 20 nanoseconds and 800 nanoseconds. The computational tasks by each app user are generated at some intervals. These intervals are selected by uniform distribution over [2 seconds, 10 seconds]. Experiments are run on the data sets by changing various parameters such as app vendors, app users, and edge servers. The data sets are summarized in Table 4.1. Each experiment is repeated 50 times, and the results are averaged. It allows neutralizing extreme cases like sparse or dense server/app user distributions. The error bar with a 95 percent confidence interval is depicted by the vertical lines in the result figures of each experiment.

4.6.3 Performance Comparison

We evaluate the performance of the ERA algorithm by conducting experiments on three types of experimental settings given in Table 4.1. In each experiment, the ERA

algorithm performance is compared with the baseline and state-of-the-art approaches in terms of the following metrics.

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

Experiments 1, 2, and 3 correspond to set-1, set-2, and set-3 given in Table 4.1. Experiment 1, 2, and 3 evaluates the algorithm's performance with the different numbers of app users per vendor ranging from 100 to 1000, the different number of edge servers ranging from 10 to 100, and the different number of app vendors ranging from 5 to 50, respectively.

4.6.3.1 Experiment 1

The experimental findings illustrated in Fig. 4.2 compare the algorithm's performance with different numbers of app users per vendor ranging from 100 to 1000, with the number of edge servers being constant at 50. The results depicted in Fig. 4.2a show

that the average cost per app vendor increases when app users increase. ERA algorithm gives a more cost-effective solution than other alternatives as it appropriately allocates resources by utilizing the multi-tenancy. This happens because ERA maximizes the number of app users of similar services on each edge server that effectively utilizes the multi-tenant edge server. Fig. 4.2b shows that the overall allocated users to edge servers decreases with increasing the number of app users as proximity and capacity constraints of edge servers may leave app users unallocated. The ERA algorithm outperforms PRDS, TPDS, MCFH, Greedy, and Best-Fit. The reason is that ERA assigns more app users of a similar service to each edge server than to others, allowing for more app users to be served on a multi-tenant edge server with similar resources. As a result, various application components, such as data and programs, can be reused, leading to faster computation without increasing the computing resources. Allocated app users per edge server also follow a similar pattern as overall app user allocation. Therefore, each edge server serves more app users under the ERA algorithm compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 4.2c. The ERA algorithm enables each edge server to serve more app users of similar services than other state-of-the-art and baseline techniques. Therefore, the number of app vendors per edge server is lower in the ERA algorithm than others, as shown in Fig. 4.2d. The experimental results in Fig. 4.2e illustrate that as the number of app users increases, so does the number of edge servers required. The ERA performs well as it motivates the app vendors to switch on edge servers used by more services subject to proximity and capacity constraints. When the number of app users exceeds a threshold, all the approaches use the approximately maximum available edge servers at some point, e.g., 700 app users in our experiments. The results illustrated in Fig. 4.2f show that as the number of app users increases, the service response time to users increases. The ERA algorithm maximizes the number of app users per server who belong to similar services, enabling app users to utilize most of the cache contents and lowering the number of swapping for application programs

and data. As a result, the resource overhead is reduced, and ERA outperforms other systems in our experiments.

4.6.3.2 Experiments 2

In this experiment, the number of edge servers is varied from 10 to 100, and the app vendors and app users per vendor are kept constant at 25 and 500, respectively. Fig. 4.3a shows how the average cost of each app vendor changes as the number of edge servers increases. As only a few edge servers are available to app users at the start (10 edge servers), a smaller number of app users receive computational services. Consequently, the app vendors' cost is less. ERA algorithm outperforms PRDS, TPDS, MCFH, Greedy, and Best-Fit as the number of available edge servers increases. The reason is that the ERA algorithm reduces the required number of edge servers per app vendor by allocating resources appropriately to maximize the usage of multi-tenant edge servers. In each of the PRA, PRDS, MCFH, and TPDS, the cost incurred by each app vendor is approximately flattened when edge servers exceed a threshold, e.g., 70 in our experiment. This happens because the number of available edge servers exceeds the required edge servers, resulting in numerous edge servers being unutilized. On the other hand, Greedy and best -Fit does not allocate resources in order, resulting in a non-optimal number of edge servers in use. Therefore, the cost of each app vendor increases with increasing edge servers in Greedy and best-Fit.

The findings depicted in Fig. 4.3b show that the number of allocated app users increases as the availability of the edge servers increases in a given geographical area. When the number of edge servers is large enough, such as 80 in our experiment, all approaches almost allocate all app users to edge servers. ERA algorithm performs well as it allocates resources to more app users of a similar service on each edge server. Due to the same reason, the ERA algorithm allocates more app users per edge server compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 4.3c. These

outcomes show that the resource utilization under the ERA algorithm is higher than the PRDS, TPDS, MCFH, Greedy, and Best-Fit. When edge servers are 10, the number of users per edge server in ERA, PRDS, MCFH, and TPDS is very high because the app vendors fully utilize a limited number of edge servers. In that case, many app users can not be allocated to any edge server. After that, the edge server utilization decreases to a certain point, e.g., 50 edge servers in our experiment, because resources on some edge servers may be allocated to only a few app users. After a certain number of edge servers in a given geographical area, such as 50 in our investigation, the number of app users per edge server increases. The reason is that when edge servers exceed 50, the ERA, PRDS, MCFH, and TPDS reorder the resource allocation in a way that abandons the resources of edge servers used by very few app users and relocates app users among a limited number of edge servers. Hence, if the number of edge servers in a given geographical area exceeds a threshold, this limits the number of edge servers in use and results in high utilization of these edge servers by a fixed number of app users. The number of app users assigned to each edge server in the Greedy and Best-fit approach follows the opposite behavior compared to the ERA, PRDS, MCFH, and TPDS. This happens as Greedy and Best-fit do not allocate computing resources to app vendors in any order.

The results illustrated in Fig.4.3d present that the number of services per edge server decreases with the increasing number of edge servers. The reason is that the app vendors can assign more app users on single edge servers as the number of choices increases. Fig.4.3d shows that the ERA algorithm makes proper use of the multi-tenant edge servers. The findings in Fig. 4.3e illustrate that, up to a point, the number of edge servers used in each technique is nearly identical, e.g., 30 in our experiments, as there is a lower availability of edge servers. As described above, the overall required edge servers are lower in the ERA algorithm. The observations in Fig. 4.3f depict that service response time decreases as the availability of edge servers increases. The ERA

algorithm works well because it maximizes cache content utilization while increasing data and software reuse.

4.6.3.3 Experiment 3

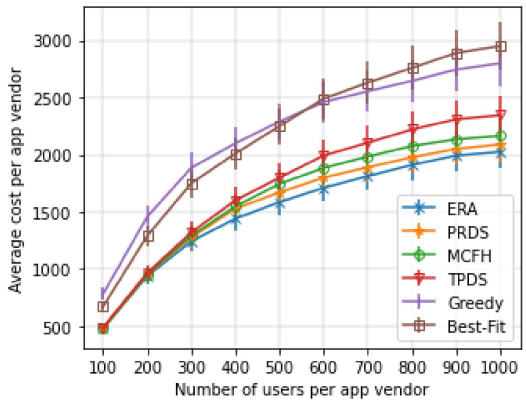
Fig. 4.4 shows the experimental results, which compare the algorithm's performance with a variation of app vendors ranging from 5 to 50, with app users per vendor and edge servers set at 500 and 50, respectively. The results depicted in Fig. 4.4a show that the average cost per app vendor decreases with an increase in app vendors as the fixed cost of the edge servers distributes among them. The solution achieved by the ERA algorithm is more cost-effective than other approaches as it allocates resources effectively to maximize the multi-tenancy use. The reason is that the ERA algorithm maximizes the number of app users of similar services on each edge server that effectively utilizes the multi-tenant edge server. Fig. 4.4b shows that the overall allocated users to edge servers decrease with app vendors. It happens because an increase in app vendors leads to an increase in app users, which causes edge servers to become unavailable owing to proximity and capacity restrictions. The ERA algorithm performs better than PRDS, TPDS, MCFH, Greedy, and Best-Fit. The reason is that ERA assigns more app users of a similar service to each edge server than to others, allowing for more app users to be served on a multi-tenant edge server with similar resources because of various application components, such as program and data, can be reused. Due to the same reason, in the ERA algorithm, each edge server can serve more app users compared to PRDS, TPDS, MCFH, Greedy, and Best-Fit, as shown in Fig. 4.4c. Compared to other state-of-the-art and baseline methodologies, the ERA algorithm allows each edge server to serve more app users of similar services. Thus, the number of app providers per edge server is lower under the ERA algorithm, as shown in Fig. 4.4d. The findings in Fig. 4.4e show that when the number of app vendors increases, the number of edge servers required increases as well. The ERA performs well as it motivates the app vendors

to switch on edge servers used by more services subject to proximity and capacity constraints. The results in 4.4f show that as the number of app vendors grows, so does the service response time to app users. The ERA algorithm performs well, as discussed above in Experiments 1 and 2.

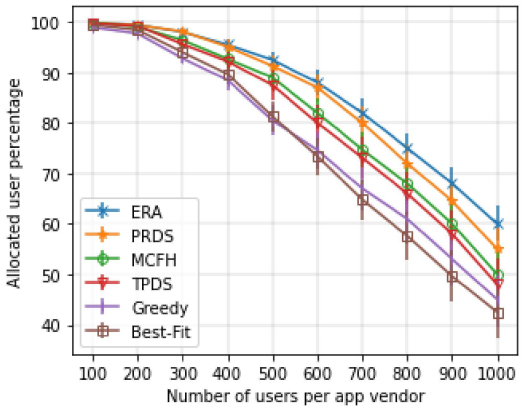
4.7 Summary

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

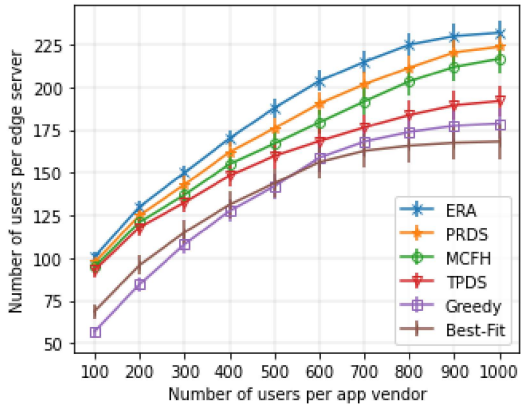
This study opened many new avenues for future investigation of the ERA problem. 1) the effects of mobility and trajectories of app users on ERAGame can be investigated. 2) Future research could look into app users' mobile participation in ERAGame, including existing app user departures and new app user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



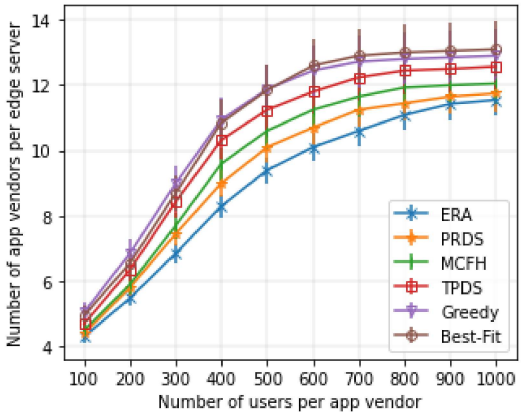
(a) Users per vendor vs. cost per vendor.



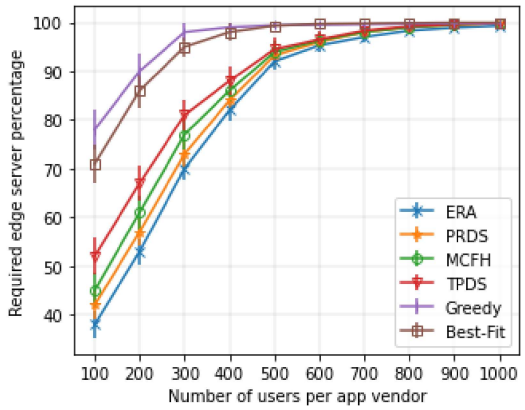
(b) Users per vendor vs. allocated users.



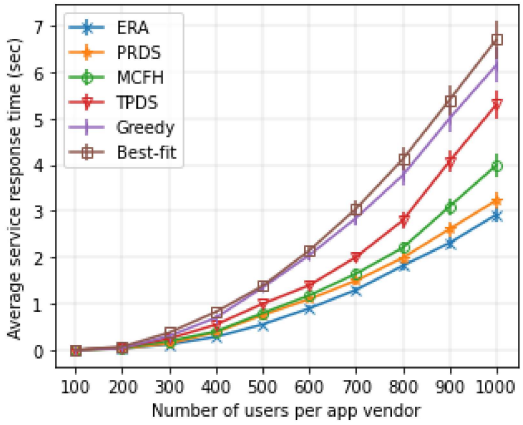
(c) Users per vendor vs. allocated users per server.



(d) Users per vendor vs. vendors per server.

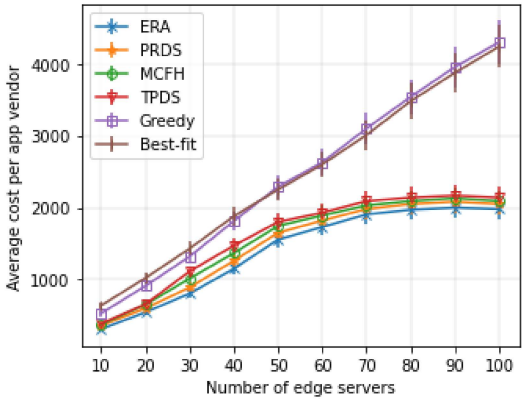


(e) Users per vendor vs. required edge servers.

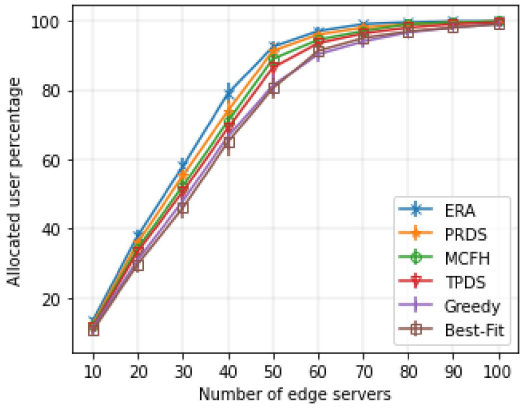


(f) Users per vendor vs. service response time.

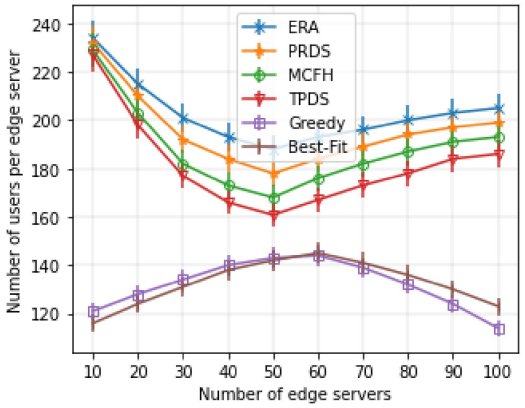
Figure 4.2: Experimental results show the performance of ERA algorithm with varying number of app users per app vendor. In all the figures vertical line represents the error bar with a confidence interval of 95%.



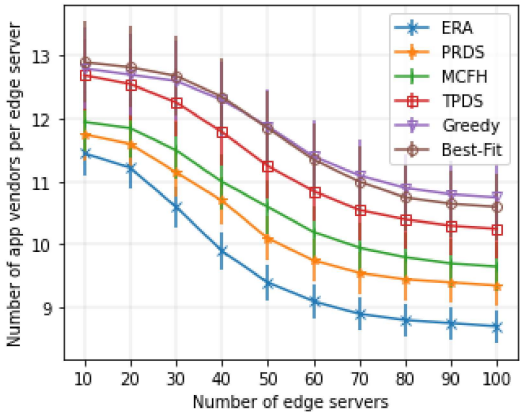
(a) Edge servers vs. cost per app vendor.



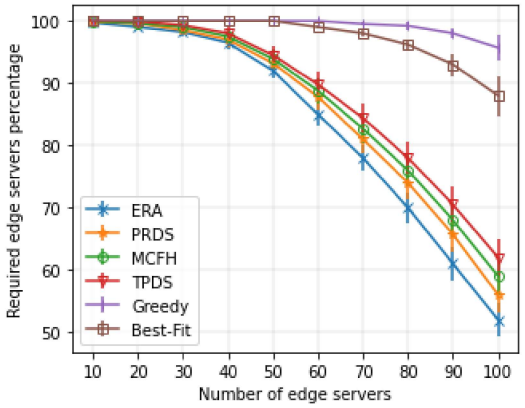
(b) Edge servers vs. allocated users.



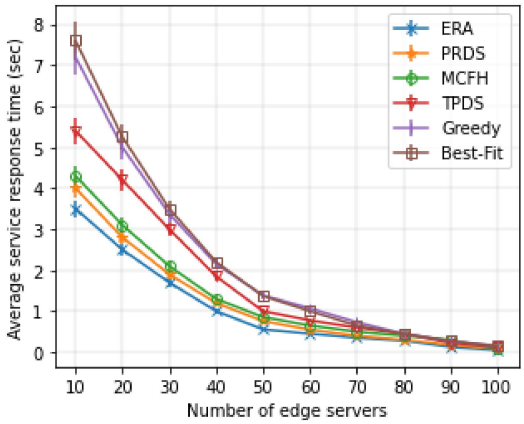
(c) Edge servers vs. allocated users per server.



(d) Edge servers vs. app vendors per server.

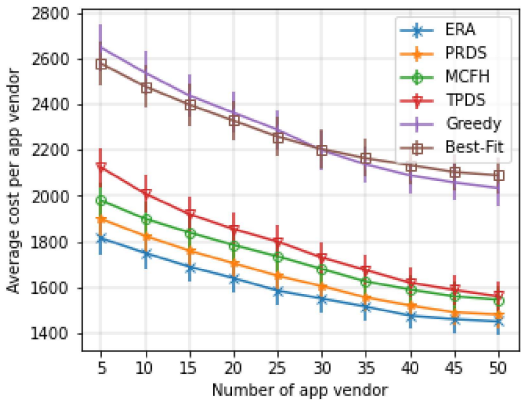


(e) Edge servers vs. required edge servers.

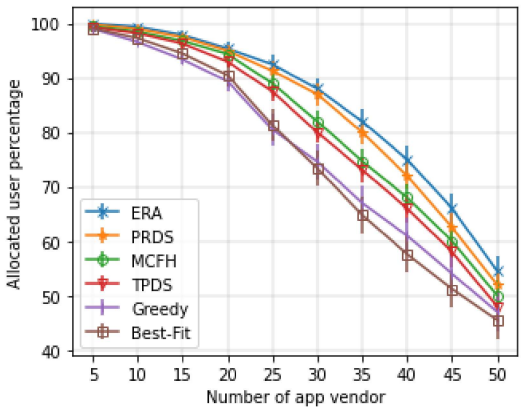


(f) Edge servers vs. service response time.

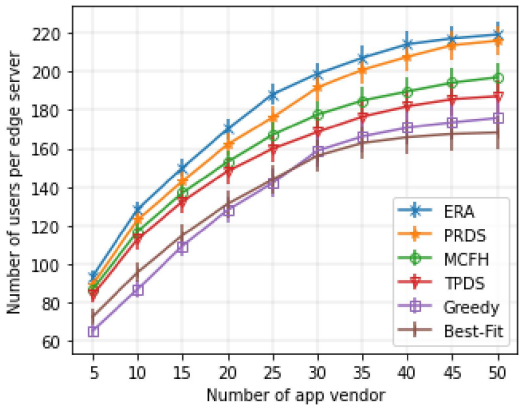
Figure 4.3: Experimental results show the performance of ERA algorithm with varying number of edge servers. In all the figures vertical line represents the error bar with a confidence interval of 95%.



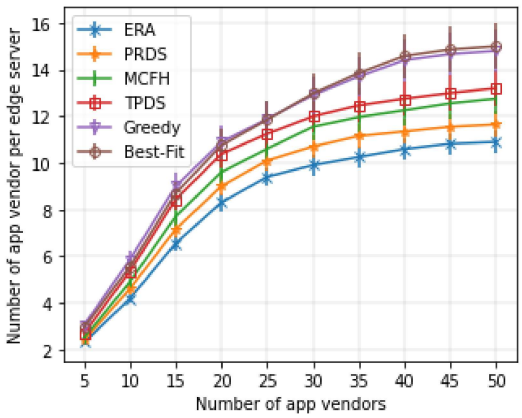
(a) App vendors vs. cost per app vendor.



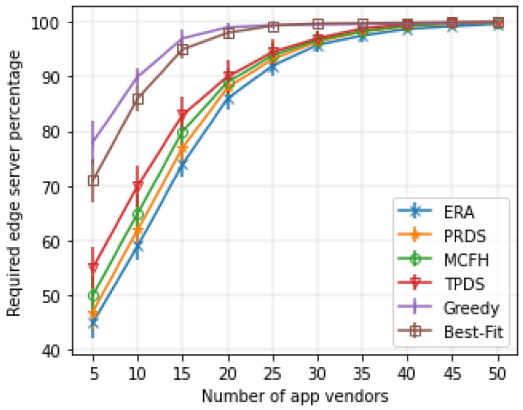
(b) App vendors vs. allocated users.



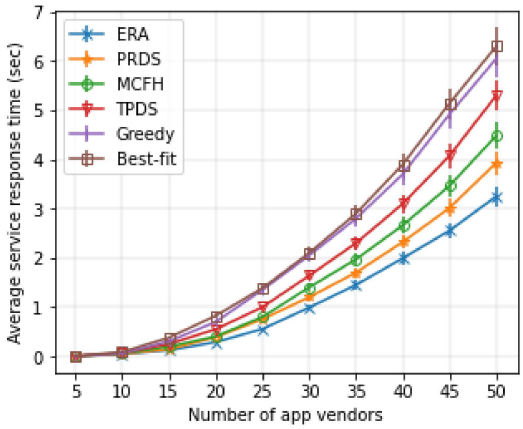
(c) App vendors vs. allocated users per server.



(d) App vendors vs. app vendors per server.



(e) App vendors vs. required edge servers.



(f) App vendors vs. service response time.

Figure 4.4: Experimental results show the performance of ERA algorithm with varying number of app vendors. In all the figures vertical line represents the error bar with a confidence interval of 95%.