Strategies for User Allocation and Service Placement in Multi-Access Edge Computing

Subrat Prasad Panda

Strategies for User Allocation and Service Placement in Multi-Access Edge Computing

Dissertation submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Computer Science

by

Subrat Prasad Panda
[Roll No: CS1913]

under the guidance of

Ansuman Banerjee
Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

July 2021

To my family and my supervisor

CERTIFICATE

This is to certify that the dissertation titled **"Strategies for User Allocation and Service Placement in Multi-Access Edge Computing"** submitted by **Subrat Prasad Panda** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Ansuman Banerjee
Professor,
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgement

I would like to convey my highest gratitude to my supervisor, *Dr. Ansuman Banerjee*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. Before pursuing this thesis, I didn't have much knowledge about the research domain or any experience of good research. He aided me to hone my reasoning skills and illuminated the ways to improve my research and writing skills. His guidance made me, what people informally call, "from zero to hero."

My deepest thanks to the faculties of Indian Statistical Institute, for their support.

I would also like to thank *Dr. Arani Bhattacharya*, Assistant Professor (CSE), Indraprastha Institute of Information Technology, Delhi, for his constructive comments and discussions on my research. Also, many thanks to *Kaustabha Ray*, a research scholar at Indian Statistical Institute, Kolkata, for his useful advice while performing experiments for this research.

Last but not the least, I would like to thank all my friends for their help and support. I thank all those, whom I have missed out from the above list.

Subrat Prasad Panda
Indian Statistical Institute
Kolkata - 700108, India.

Abstract

The emergence of Multi-access Edge Computing (MEC) grants service providers the ability to deploy services at edge servers near base stations to mitigate the effects of high network latencies often encountered in cloud-based system deployments. As users move around, their application service invocations are routed to proximate MEC servers en route to curtail the high latencies of cloud communication networks. In contrast to cloud servers, edge servers have constraints on resources such as computation, storage, energy, etc. Embedded devices often function as edge servers which are quite less flexible and resource impaired when compared to their full-fledged cloud server counterparts when hosting services. Thus, placement and allocation of services on edge servers and binding user service requests to the service instances hosted on the edge pose a number of research challenges. Also, the movement of users in the edge environment leads to the challenge of migration of service data and placement of hosted services. To efficiently use the available edge server resources and handle the mobility of users, an edge user allocation policy is designed. An edge user allocation policy determines how to allocate service requests from mobile users to MEC servers. An efficient edge user allocation policy is quite challenging to design due to the influence of an extensive variety of factors like the mobility of users, considerations of optimal Quality-of-Service (QoS) and Quality-of-Experience (QoE), variable latencies, stochastic nature of user service requests, limited resources, device energy constraints and so forth.

This thesis predominantly focuses on the user allocation and service placement problems in MEC with an objective to provide efficient and scalable solutions. Classical MEC policies that bind user service requests to edge servers, seldom take into account user preferences of QoS and the resulting QoE. In our first contribution, we propose a novel user-centric optimal allocation policy considering user QoS preferences, with an attempt to maximize overall QoE. Furthermore, traditional allocation and placement policies cater to service request allocation and placement without much consideration of workload fluctuations. To address such issues, the second contributory chapter of this thesis proposes a variation aware stochastic model for user service allocation. In addition, current state-of-the-art techniques assume MEC resource utilization to be linearly dependent on the number of service request demands and usages, i.e. the combined resources utilized by a group of user services is the sum of service resource utilization per user. In our third contributory chapter, we propose a real-time on-device learnable Reinforcement Learning (RL) framework to design user allocation policies that accommodate the non-linear nature of resource utilization by services.

We implemented our proposed approaches on real-world datasets and analyzed the performance of our proposed algorithms to demonstrate the efficiency of our proposals. We believe our work will open up a lot of new research directions and applications of learning based methods in the MEC context.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rapid compounding growth of connected mobiles and Internet of Things (IoT) devices has led to a manifold increase in the number and sophistication of mobile software services and applications [21]. Real-time application services like Augmented Reality (AR), Virtual Reality (VR), online gaming, video processing for autonomous/connected vehicles and so forth require significant computational power and in turn, lead to high battery and energy consumption when performed on the devices [21] [31]. To mitigate these bottlenecks, devices are usually complemented with cloud services to enhance the QoS/QoE metrics of application services [11] [18] [73]. However, such a mechanism does not always necessarily conform to QoS requirements of latency-sensitive services [21] [31] [58]. The distant cloud servers affect the service latencies due to the hopping of network data packets through many intermediate devices. Also, the costs involved in the development and maintenance of cloud infrastructure makes it non-viable for omnipresence. MEC [10] represents a promising new paradigm in which the computing devices or edge servers provide compute-intensive low-latency services by being installed much closer to the user than cloud data centres. MEC servers are present today in cellular towers, mini data centres or even in homes of mobile users.

MEC allows service providers to deploy services on MEC servers located near base stations with low latency access, consequently bringing cloud-based storage, computation, measurement, and management more adjacent to the end-user to ensure QoE, optimize resource use, eventually generating revenue for network operators [29]. When service requests are generated from user mobile devices, instead of executing them on the resource-deprived devices or the distant cloud, the services requests are offloaded to nearby MEC servers installed at mobile base stations. MEC servers spawn containers for executing services, in case service containers are unavailable, they get fetched from another MEC server or the cloud server. The service requests are met by containers hosted on the MEC servers saving the latency of communication to a cloud server. After the service requests get fulfilled, the service containers are cached for later demands and eventually discarded if unused. The increasing demand for compute-intensive low latency applications, like real-time vehicle identification, object detection and route prediction is gradually leading to increased adoption of MEC and installation of MEC servers [2]. By 2024, 5G is expected to be a multi-million-dollar industry with enterprise deployments [71]. Several research papers have referred to MEC for application in Healthcare, Video Analytics, Big Data Analytics, Connected Vehicles, Smart Grid, Wireless Sensor Networks etc. [4, 36, 43, 62]

The MEC paradigm certainly solves service latency issues, but inevitably is not a panacea. It is tangled with many difficult challenges which hinder real-world implementation. It has garnered significant research attention to resolve some of these technology implementation challenges that have otherwise impaired its widespread adoption. Major challenges include computation offloading, user allocation, service placement, user mobility, resource allocation, energy, security etc. Recently, Machine Learning

(ML)-based solutions to some of the MEC problems are on the rise. Studies in the MEC context mostly propose algorithms for one or a combination of challenges and provide heuristic, approximation or learning-based solutions.

User Allocation and Service Placement in MEC are two prominent aspects dealt with in this thesis, which aim to determine the user-service-server binding policy for the routing of service requests i.e. which service requests from which users are provisioned by which MEC servers in their vicinity, as they move around. The Edge User Allocation (EUA) problem aims to ensure users are allocated edge server resources while satisfying constraints on coverage, resource availability, temporally varying service requests, mobility of users, varying resource footprints for service execution, latency requirement, and so on. Several algorithms have been proposed to solve the EUA problem [26, 28, 30, 31, 47, 48, 63]. These proposals typically attempt to derive effective edge user allocations, while optimizing one or more of the following metrics like latency, count of allocated users, energy, etc.

A service placement policy, on the other hand, determines which service containers should be deployed on which MEC server [45, 46, 49, 70]. Deploying every service on the edge server is infeasible as it will needlessly consume the resources of edge servers. Many times, edge servers are unequipped with hardware resources to handle the diverse requirements of services. For example, an edge server not equipped with GPU hardware cannot efficiently manage modern gaming or vision workloads in an effective way. Therefore, service placement policies dictate which services are to be hosted on which edge server. Evidently, provisioning all services on all servers is a wasteful proposition being agnostic to actual service usage and request needs.

In this thesis, our key objective is to address the user allocation and service placement problem with proposals of heuristic and learning-based algorithms which can be used efficiently in a real-world scenario. We believe this work will motivate research towards ML-based solutions to the MEC challenges.

Apart from the challenges addressed in this thesis i.e. user allocation and service placement, other challenges that are widely researched in MEC are enlisted below:

(a) *Computation Offloading:* Computation offloading represents the process of migrating computing tasks to external sources. The external sources are servers, like edge servers or cloud servers, capable of handling offloaded service requests [34]. Unlike a cloud server, edge server resources have limited capacity, so offloaded service requests could encounter latency problems due to congestion at the edge server. As computation offloading requires energy and network bandwidth, whether to offload a service to an external server instead of executing it on the device itself is an intriguing challenge. Offloading policies determine what/when/how to offload workloads from handheld devices to the cloud or edge server [16, 18, 26, 58, 60].

(b) *Mobility:* MEC environments have to take care of the movement of mobile users in and out of the coverage of edge servers. A migration policy handles the movement of users from coverage of an edge server with an aim of suffering minimal lags and less service interruption. The policy determines how/where to move the service requests and already processed service data [16, 47, 64, 77].

(c) *Resource Allocation:* The edge server resources are scanty in comparison to cloud servers. So, edge resources need to be utilized scrupulously to achieve better efficiency [31, 78].

(d) *Energy:* Offloading and executing services on the edge server consumes device battery and energy. Sometimes, offloading services to edge servers consume more energy than executing services on mobile devices. User allocation and offloading policies are obtained to reduce the energy footprint due to offloading of services to the edge servers [8, 17, 42, 64, 74].

(e) *Privacy and Security:* MEC servers are generally installed near the unsecured base station. Ad-

versaries can break into MEC servers to expose services/data of mobile users which may hinder the privacy and security of the users. MEC policies need to maintain the confidentiality, integrity and privacy of the mobile users [4, 51].

Due to the advancement in ML research and availability of GPU inside edge devices, the application of ML techniques to MEC challenges is getting widely adopted. ML techniques like Deep Reinforcement Learning (DRL), Deep Learning (DL), etc. are extensively used today to model the system or for solving the optimization problems in the MEC context [6, 14, 21, 25, 32, 36, 60, 65–68].

1.1 Motivation of this dissertation

The EUA problem is inherently challenging due to the involvement of a multitude of attributes. The rise in investment for the deployment of MEC servers and attention from service providers to enhance user experience paves the way to development of algorithms for use in the real-world setting. Current proposals in MEC predominantly focus on the development of computationally fast algorithms that can be used online in real-time, so heuristic or approximation algorithms are sought after. With the advent of Machine Learning, research is moving towards learning-based approaches to combating the EUA problem.

A recent work [31] used qualitative QoS level offerings by service providers in designing the static EUA policies. Static policies do not handle the migration of users from the coverage of edge servers. This work, however, does not consider a user's QoS preferences when deciding user-server bindings. User QoS preferences are dynamic and are influenced by network bandwidth, energy constraint, etc. For example, a user with a depleted battery might be interested in services with low QoS.

The work in [49] investigated the problem of joint optimization of service placement and allocation and proposed algorithms to demonstrate the benefits of an orchestrated solution that deals with these two crucial MEC steps together. Indeed, as the reported experiments indicate, the joint objective optimization approach outperforms many others, which deal with piecemeal objectives. However, this approach as well, and in fact, most of the work in MEC literature, cater to service allocation and placement without considering actual resource utilization of services on edge servers during execution, and thus often fall short of providing the expected performance guarantees.

Moreover, most of works in MEC literature model the MEC system assuming linear dependence of resource utilization on the number of service requests hosted on the edge server. In reality, however, the resource utilization by services does not scale linearly when the number of requests grows as shown in [27, 37, 51, 53, 61] using the Google cluster trace dataset [54]. The non-linearity arises due to the effect of various internal system attributes such as software/hardware architecture, operating system, number of cores, varying nature of service workloads in CPU/GPU, service invocation pattern, etc., which are often ignored in prior research work.

The primary motivation of this thesis is to develop scalable and efficient user-centric heuristic solutions to EUA problems. Specifically, this thesis has the following motivations:

- We believe the designing of allocation policies should be user-centric and needs to be adaptive to user demands and preferences. The policy should be scalable and usable in a real-world edge environment.
- Use of real-world data to model the non-linear system dynamics with ML is increasingly becoming a better alternative, instead of modelling the system dynamics mathematically which

is often challenging. We aim to explore the advantages of learning based user allocation policy design over traditional methods.

1.2 Contributions of this dissertation

The objective of this thesis is to design scalable, user-centric and data-driven algorithms to solve the EUA problem in the MEC context. In particular, this thesis proposes new heuristics and a DRL based algorithm for the design of user allocation policies. The contributions of this thesis are briefly described below:

- *User Allocation with User Specified QoS Preferences*: User allocation policies seldom take into account user preferences of QoS and the resulting QoE. Most of the research work focuses on service provider-oriented algorithms to maximize the profit to service providers. Consequently, service preferences of edge users go unaccounted for, and users are forced with service QoS regulated by service providers. We propose a novel user-centric allocation policy considering the QoS preferences of users to maximize the overall QoE.
- *Service Allocation and Placement with Workload Fluctuations*: The user centric policies need the attributes of the user's preference for designing policies. Asking users their QoS preference each time while requesting a service from an edge server is cumbersome, also, it may hinder the user's experience to enjoy the requested service. User QoS preferences are highly dynamic, so predicting QoS preferences for each user automatically is very challenging. However, QoS preferences of users directly affect the computational/network resource utilization on the edge server, thereby, resulting in different resource utilization footprints than estimated during service execution due to workload fluctuations. Apart from a user's QoS preference, numerous other reasons affect service workload fluctuation like co-located service request workloads from other users hosted on the same edge server, the scheduling algorithms adopted on the edge server, server dynamics, hardware configurations, temperature, etc. So, instead of predicting the QoS preferences of each user or urging the user to specify their preferences, we propose a variation aware stochastic model which considers workload fluctuations to obtain the user allocation and service placement policies. Traditional allocation and placement policies, to the best of our knowledge, cater to service request allocation and placement without much consideration of workload fluctuations.
- *User Allocation using Deep Reinforcement Learning*: Current state-of-the-art techniques assume the total resource utilization on an edge server is equal to the sum of the individual resource usages of service requests being served from an edge server. However, the relationship between resources utilized on an edge server with the number of service requests to the edge server is usually highly non-linear. Moreover, in our proposal of the stochastic approach to handle workload fluctuations, we assume the service resource utilization to follow certain distributions. Mathematically modelling such highly non-linear systems for resource utilization is challenging. We follow a real-world data-driven approach using DRL to model resource utilization of services on the edge server. We utilize this learned model to design user allocation policies.

1.3 Organization of the dissertation

This dissertation is organized into 6 chapters. A summary of the contents of the chapters is as follows:

Chapter 1: This chapter contains an introduction and a summary of the major contributions of this work.

Chapter 2: A detailed study of relevant research is presented here.

Chapter 3: This chapter describes a heuristic algorithm to obtain user-centric user allocations.

Chapter 4: This chapter describes a workload fluctuation aware stochastic model to determine a service allocation and placement policy.

Chapter 5: This chapter describes a DRL based approach for design of a user allocation policy.

Chapter 6: We summarize with conclusions on the contributions of this dissertation.

Chapter 2

Background and Related Work

In this chapter, we present a few background concepts related to the user allocation and service placement problem. We subsequently present a brief overview of prior works proposed on the problem.

2.1 Preliminary Concepts

We explain the necessary concepts like MEC architecture, EUA problem, Integer Linear Programming, Cantelli's inequality, and RL in this section.

2.1.1 MEC Architecture

Figure 2.1 shows a typical MEC architecture. Mobile devices like smartphones, intelligent vehicles, intelligent drones, etc. generate service requests for computationally intensive applications. The request is then redirected to nearby MEC servers installed at Base Stations (BS). The base stations and the cloud servers are connected to the core network. In case, MEC servers are incapable of processing the request, the service requests are routed to cloud servers with added latencies. Whenever a service request gets assigned to a particular server, the server spawns required service containers to serve the user who generated that service request. Additionally, if a service container is unavailable on the MEC server, it gets fetched from the cloud server and cached for further use.

2.1.2 EUA Problem

The Edge User Allocation problem is to find optimal user-service-server binding given a set of users and a set of edge servers with their location and resources available. The illustration in Figure 2.2 shows an example scenario of a typical MEC environment. Users $u1, u2 \dots u7$ are requesting services hosted on the edge servers $e1$ and $e2$. Edge users should be inside the coverage radius of the edge server to receive services hosted on it. Users $u1$, $u2$, $u3$ are under the coverage of edge server $e1$, users $u6$ and $u7$ are under the coverage of $e2$, whereas, users $u4$ and $u5$ are under the coverage of both the servers $e1$ and $e2$. The user allocation problem aims to determine user-server bindings to achieve optimal performance in the MEC environment. For this example scenario, as users $u4$ and $u5$ can be connected to both the edge servers, the allocation policy needs to determine the binding to only one of the servers for each user.

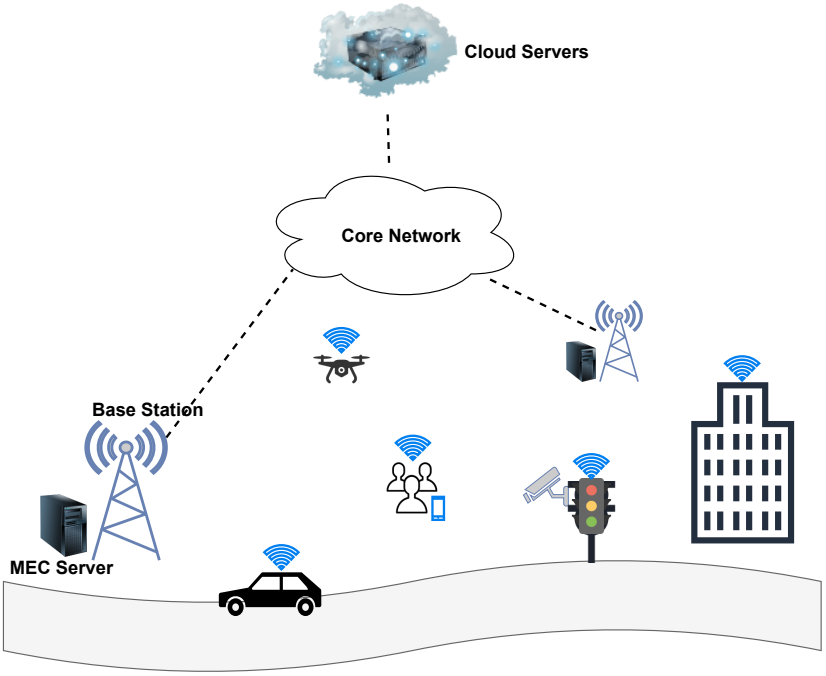


Figure 2.1: MEC Architecture

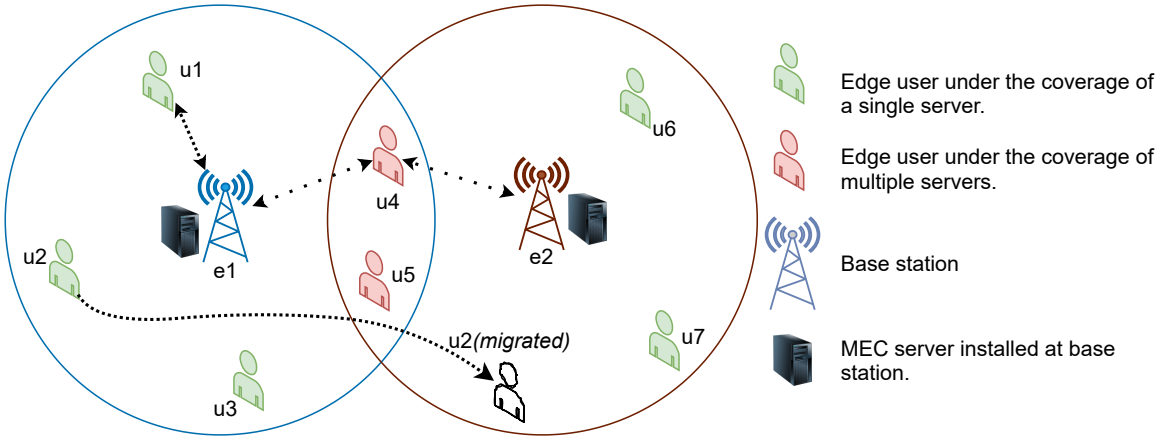


Figure 2.2: EUA Problem in MEC

2.1.3 Integer Linear Programming (ILP)

An ILP [57] is an optimization technique that enforces all variables to be integers and the objective function and the constraints to be linear. The objective of ILP is to optimize a given linear function while satisfying a set of linear constraints. We now illustrate an ILP formulation using the following example.

Example 2.1 *Consider we are given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Our objective is to identify a minimal vertex cover, where the vertex cover of a graph refers to a set of vertices such that each edge of the graph is incident on at least one vertex of the set. In order to formulate the ILP, we first define a decision variable x_i corresponding to each vertex of the graph.*

$$x_i = \begin{cases} 1, & \textit{if vertex } v_i \in V \textit{ is considered in the vertex cover set} \\ 0, & \textit{Otherwise} \end{cases}$$

Our objective is to minimize the number of vertices chosen for the cover to get the minimal cover, i.e.,

$$Minimize \sum_{v_i \in V} x_i$$

Additionally, we have the following constraint representing that at least one vertex of each edge belongs to the vertex cover set.

$$\forall (v_i, v_j) \in E, \qquad x_i + x_j \geq 1$$

■

2.1.4 Cantelli's Inequality

If X is a random variable with mean μ and variance $\sigma^2 (\neq 0)$ and t is a positive real number, one-sided Chebyshev's inequality (Cantelli's inequality) can be stated as:

$$P[\frac{X - \mu}{\sigma} > t] \leq \frac{1}{1 + t^2}$$

2.1.5 Reinforcement Learning

In this section, we provide a brief overview of RL. We initially introduce the concept of Markov decision processes (MDPs) and then discuss the use of RL to solve them [59].

MDP is a stochastic mathematical model that gets adopted in scenarios that rely on taking decisions. It can solve a variety of optimization problems. An agent in MDP is the decision-maker who decides the action at each step, and a reward is received accordingly. MDP is defined by a five element tuple *(State, Action, Policy, Reward, Discount Factor)*, where:

1. State: Set of parameters used to describe the current state of the agent.
2. Action: Set of actions that an agent can take to go from one state to another.
3. Policy: The policy dictates which action should be taken by an agent at any particular State.
4. Reward: The reward achieved by an agent due to the decision of opting for a particular Action for a State.
5. Discount Factor: The factor which describes how much future reward will affect the present decision for a certain action.

The goal of MDP is to obtain an optimal policy for the agent such that it can achieve the best reward at each state. Model-based RL methods typically work on an MDP, while the model-free variants of RL are also available. In Deep Reinforcement Learning (DRL), a deep neural network is utilized as the RL agent to learn an optimal policy for an MDP. Specifically, the attributes of a state in the MDP are input to a deep neural network agent and an optimal action or policy is output from it.

2.2 Related Work

In this chapter, we present a brief overview of various policies proposed in literature on the edge user allocation and service placement problems in the MEC context i.e. the policies that determine user-service-server binding. In addition, we discuss studies on the application of ML and DRL to MEC challenges. Prior related works can be categorized into three key categories:

- studies focusing majorly on proposing different algorithms to solve the user allocation and service placement problem
- works which have used ML techniques to model the performance of cloud/edge
- works that have used DRL to solve resource allocation or optimization problems

User Allocation and Service Placement Problem:

Several works formulate EUA as an optimization problem and use a variety of techniques like ILP, approximation algorithms, or heuristics to solve them efficiently [6, 22, 30, 31, 46, 49, 66, 67, 74]. For example, [30] and [31] formulate the allocation problems as a version of the bin-packing problem, with the objective being to maximize the number of users allocated to the edge or the QoE of users. Authors in [74] propose optimal and approximate mechanisms for allocating network resources in MEC. In [19], user allocation is done by a game-theoretic approach.

The work [49] formulates joint allocation and service placement as that of minimizing the number of users allocated to each cloud server and demonstrate the effectiveness and efficiency of approximation algorithms in both static and dynamic contexts. In [46], the authors derive an approximation algorithm by incorporating rewards whenever a user's requirements for resources are honoured. In [22], the authors formulate a time-slotted model and develop a polynomial-time approximation by jointly optimizing service placement and request scheduling, i.e., which user requests are to be routed to which edge server with services deployed. The work in [64] considers minimizing each edge server's energy consumption as an optimization metric and consider minimizing each MEC server's energy consumption.

Service Placement has been considered by several authors in both static [46], dynamic [45] service contexts. The work in [48] additionally considers data transfer and availability for making placement decisions. In [15], the authors also consider base-stations collaborating to make service placement decisions.

In [6, 66, 67], the authors develop a mathematical model of an edge system to solve the optimization problem using reinforcement learning. Although DRL is used for the EUA problem in [32], it nevertheless assumes a linear relationship between resource utilization and execution time.

ML-based Performance Modelling:

Some works utilize machine learning-based performance models to predict the service attributes for different cloud architectures. For example, PARIS [72] and CherryPick [7] identify the best Virtual Machine (VM) for different workloads using random forests and Bayesian optimization respectively. In SLAOrchestrator [44], a linear regression technique is used on a service workload footprint dataset to predict workload performance.

The work in AutoPilot [55] applies a multi-armed bandit technique, a RL method, to identify an action to scale up or down execution on cloud systems. The work in [5] uses a deep neural network

to learn the system dynamics of LTE Network devices to allocate users to different base stations by recording real-world datasets over a significant period.

Deep Reinforcement Learning based Solutions:

Many systems utilize DRL, a DL-based RL technique, to optimize their performance, although not necessarily for allocation of users to cloud or edge [33]. For example, Pensieve [35] uses DRL to allocate bitrates to each video streaming client. The work in [69] allocates channel bands for transmission to IoT devices using DRL. In [75] authors use DRL to allocate power to different antennas. The work [40] uses DRL to perform accurate indoor localization of users using Bluetooth Low Energy (BLE) signals.

Finally, in the context of MEC, DRL has been used for caching data close to the users [79] and even computation offloading [14, 32]. In particular, [32] uses DRL to solve the optimization formulation instead of the conventional optimization method, while assuming a linear relationship in mathematically modeling the MEC system. Also, the work in [65] uses a Sequence-to-Sequence (S2S) neural network with DRL training to solve the problem of task offloading in MEC. These works are based on the observation that simplistic models often fail to accurately take into account the relationship between resource availability and performance in actual systems.

In this thesis work, we propose a number of novel user-centric heuristic allocation proposals. Subsequently, we propose a workload fluctuation aware allocation policy. Finally, we leverage the application of DRL to solve the EUA problem efficiently compared to the state-of-the-art. To the best of our knowledge, this is the first work that learns the relationship between resource utilization and edge server system performance using DRL to predict the number of users that can be allocated to a particular edge server. We believe that this work presents a new direction to advance the use of DRL in MEC for real-world deployment.

Chapter 3

User Allocation with User Specified QoS Preferences

Quality of Experience (QoE) is a measure of the satisfaction of a user's experiences with a service. Moreover, Quality of Service (QoS) provided to the users has a direct impact on their QoE. The overall QoE of users is a salient optimization metric to obtain allocation policies for the EUA problem. A recent work [31] has demonstrated the quantitative correlation between service QoS level offering by service providers with the overall QoE of edge users. This work has additionally shown the existence of thresholds, beyond which, QoS values no longer affect a user QoE. The authors have proposed a novel view of considering the overall QoE as an optimization metric to assign the QoS level at which users will be served and obtaining the user-server binding policies. This proposal, however, does not consider a user's QoS preference level when deciding these bindings. Moreover, the binding is static, which means, after determining the allocation for a user service invocation to a specific QoS level at an edge server, the user is continued to be served at the same QoS level throughout. A static allocation is oblivious to the fact that the user may not be in a state to enjoy services at a higher QoS level all the time due to battery or other constraints. Also, the policy is not self-adaptive to consider the joining, leaving, migration or QoS preference changes of users. Our motivation in this chapter is to design a dynamic self-adaptive allocation policy that can address these variations.

The stochastic nature of service invocation patterns and the significantly large configuration space of user-service-server binding possibilities make it difficult to design an allocation policy that considers user preferences of QoS levels. In our view, allocation policies in literature are more catered towards the perspective of service providers [30, 31], aiming to optimize quantitative metrics, often ignoring users' qualitative preferences of QoS levels when making allocation decisions.

QoS levels typically have a monotonically increasing effect on the resource consumption of mobile devices and edge servers, at the edge server where the service gets executed, and at the mobile devices where wireless communication occurs for data transfer. Given the limited capacity of mobile devices, a user may not always have the requisite resources to consume a high-quality service. Consider an online gaming platform [30]. A user may express a preferred choice of resolution for rendering the game

---

This work is published as:

- Panda S.P., Ray K. and Banerjee A., Dynamic Edge User Allocation with User Specified QoS Preferences. In proceedings of the 18th International Conference on Service Oriented Computing (ICSOC) 2020, Dubai, pages 187-197.

graphics instead of using the highest possible rendering quality offered by vendors. An allocation policy that assigns the highest QoS levels available, in this case, a higher resolution, may be detrimental to the user since rendering the game at higher resolutions, will result in higher bandwidth consumption and battery depletion. As a result, policies like [31], being user preference agnostic, may allocate high QoS levels to users leading to an added aggravation. In such scenarios, a service provider may also suffer degradation in throughput as high QoS levels translate to more resource utilization at the edge servers, which could have been otherwise allocated for other users. An overly aggressive user-agnostic QoS allocation can cause new service requests being needlessly denied service in the worst case.

Our proposal in this chapter is a novel user-centric service level agreement specification that caters to both user and service providers. Specifically, we propose a novel dynamic allocation paradigm where we solve the edge user allocation problem considering individual user QoS preference levels to optimize the overall QoE of users with awareness of user's mobility and changing QoS preferences. Generally, the QoS preference of users changes with time. For example, a user initially with a high battery level on the mobile device may prefer to stream services at a high QoS level, but sometime later may choose to downgrade the QoS preference depending on the battery conditions to ease energy consumption by data communication. Our service allocation policy takes into account such user-specified adjustments in an attempt to maximize the overall user experience.

In this chapter, we formulate the problem of dynamic QoS preference aware edge user allocation and propose an ILP formulation to obtain an optimal allocation. Further, we propose a heuristic algorithm that produces near-optimal QoE allocations. We use the EUA dataset [28, 30, 31, 47], a real-world dataset as edge server locations. Additionally, we use the PlanetLab and Seattle Latency dataset [80] to generate latencies representative of MEC environments to validate our approach. Experiments demonstrate the efficacy of our heuristic algorithm to produce near-optimal allocations. We compare our results with two state-of-the-art approaches, the proposal in [31] which considers QoS and QoE, but is a mostly static solution, and a dynamic mobility aware one [47] and show our proposal outperforms both in respect of the QoE metric.

The rest of this chapter is organized as follows. Section 3.1 motivates the problem context with an example. Section 3.2 proposes the ILP model. Section 3.3 proposes our heuristic, while Section 3.4 presents the results. Section 3.5 concludes this chapter.

3.1 A Motivating Example

In this section, we present a motivating example to explain the problem context. Consider the scenario demonstrated in Fig. 3.1. There are six edge users u_1, u_2, u_3, u_4, u_5 and u_6, and two edge servers E_1 and E_2. The coverage area of a particular server within which a user requests for services is marked by a circle. For example, u_1 can only access the services from E_1, whereas, u_4 can access the services hosted at both E_1 and E_2.

A resource vector $\langle vCPUs, RAM, storage, bandwidth \rangle$ represents the resource capacity of each server [31], where $vCPU$ denotes the number of virtual CPUs. For the example scenario, assume the resource capacities of servers are denoted by vectors $s_1 = \langle 16, 32, 750, 8 \rangle$ and $s_2 = \langle 16, 16, 500, 4 \rangle$. Every server can host services at various QoS levels. Providing a service at a particular QoS level consumes a certain amount of server resources.

We assume both E_1 and E_2 host a service $\mathcal{P}$ with 3 QoS levels W_1, W_2 and W_3 as in Table 3.1. Each QoS level utilizes resources on the edge server represented by a 4-element resource vector $W = \langle vCPUs, RAM, storage, bandwidth \rangle$ and an associated QoE value. Here, W_3 is the highest QoS level. A user requesting service $\mathcal{P}$ specifies a desired QoS level from W_1, W_2 or W_3 at which the user

desires to experience the service. Furthermore, the user provides a lower tolerance threshold QoS level associated with the service request, below which provisioning of services is unacceptable.

Table 3.2 shows the initial QoS preferences of the users. In the scenario demonstrated in Figure 3.1, u_3 follows the trajectory as depicted by the curved line while all other users remain stationary. At time $t = 0$, demarcated by a black rectangle, u_3 requests the service $\mathcal{P}$ with QoS preference as W_3. Simultaneously, at $t = 0$ other users u_1, u_2, u_4, and u_5 also request for the service $\mathcal{P}$. The user u_6 is idle and does not request for any services initially at $t = 0$, but at $t = 5s$ u_6 joins and requests for $\mathcal{P}$. After user u_3 moves inside the coverage area of server E_2, at $t = 5s$, u_3 downgrades its QoS preference from W_3 to W_2, at the point indicated by the blue diamond.

QoS Level	Resource Requirement	QoE
W_1	$\langle 2, 2, 10, 1 \rangle$	1.5
W_2	$\langle 4, 4, 15, 1.5 \rangle$	4
W_3	$\langle 8, 4, 20, 2 \rangle$	5

Table 3.1: Available QoS levels

User	QoS Level	QoS Min	Allocation t=0s		Allocation t=5s	
			[31]	Our proposal	[31]	Our proposal
u_1	W_1	Any	E_1, W_2	E_1, W_1	E_1, W_3	E_1, W_1
u_2	Any	Any	E_1, W_2	E_1, W_2	E_1, W_2	E_1, W_3
u_3	W_3	W_2	E_1, W_3	E_1, W_3	E_2, W_3	E_2, W_2
u_4	W_2	Any	E_2, W_3	E_2, W_2	E_1, W_2	E_1, W_2
u_5	W_3	W_2	E_2, W_3	E_2, W_3	E_2, W_3	E_2, W_2
u_6	W_1	Any	Idle	Idle	NA	E_2, W_1

Table 3.2: User QoS details

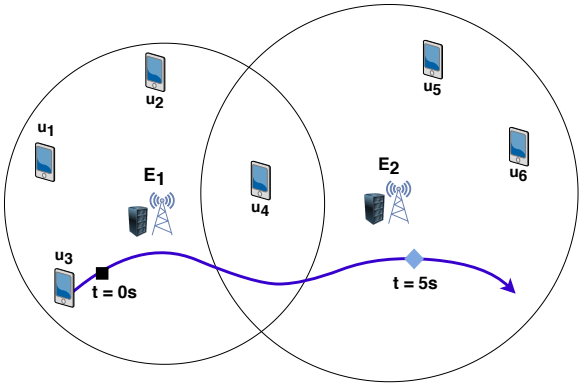


Figure 3.1: Representative MEC Scenario

3.1.1 User QoS Preference Agnostic Allocation

A user preference agnostic policy such as [31] does not consider the QoS preferences of users to generate an allocation policy. The user-agnostic policy will assign QoS levels to maximize the total QoE of all users, whether or not the user wants the assigned QoS level. The allocation is presented in Column 4 of Table 3.2 as E_k, W_p pairs indicating the edge server E_k and the QoS level W_p to which the user u_i is bound.

As illustrated in Table 3.2, user u_1 requested a QoS level W_1, however, a user preference agnostic policy allocated a higher QoS level W_2 to the user. Moreover, at $t = 5s$, this policy continues to provision u_3 at W_3 as shown in Column 6, agnostic of the fact that u_3 had requested for a downgrade to W_2. So, the user u_3 suffers added latency due to data transmission overload as the actual requirement of bandwidth is $1.5Mbps$ in QoS level W_2, but the user receives the service at QoS level W_3 with required bandwidth $2Mbps$. Also, at $t = 5s$, when u_6 invokes the service, E_2 no longer has the needed resources to serve him, considering its serving capacity and the resources already consumed. Given the coverage constraint and the locations shown, u_6 cannot be served by E_1. However, had u_3's QoS level been reduced to W_2 when u_3 changed its preference level, u_6 could be onboarded at E_2.

3.1.2 Proposed User Centric Approach

The user-centric allocation policy proposed in this chapter considers the preferences of users. As depicted in Table 3.2, the user preference aware policy attempts to allocate each user to the desired QoS level. Further, at time $t = 5s$, when u_3 indicates its change of preference level, the proposed user-centric policy reduces the QoS level allocated from W_3 to W_2. In such a scenario, it prevents the user from transmission overload as the requirement of bandwidth $1.5Mbps$ is well-taken care of.

The assignment of QoS level W_2 to user u_3 makes QoE value of u_3 to 4, which is lower than the received QoE if QoS level W_3 was assigned. However, as user u_3 requested the QoS level W_2, we consider the corresponding QoE value is good enough for the user. Additionally, since a lower QoS level corresponds to lower resource consumption at the server, we can redistribute the resources to better serve other users. u_6 can now be onboarded at $t = 5s$.

The example illustrates a comparison between QoS preference agnostic and QoS preference aware allocation showing the trade-off between resource consumption, latency and QoE. The latter is challenging to design, considering time-varying user QoS requirements while catering to user mobility. To the best of our knowledge, this is the first work towards mobility-aware dynamic user allocation with user QoS preferences.

3.2 System Model and ILP Formulation

In this section, we first formalize the MEC system model for user-centric allocation. We consider a discrete time-slotted model [47]. We denote by $U^t = \{u_1, u_2 \ \dots \ u_n\}$ the set of active users and by $S^t = \{s_1, s_2 \ \dots \ s_m\}$ the set of active edge-servers at time t. The capacity vector C_j^t $\langle CPU, RAM, storage, bandwidth\rangle$ in that order denotes the available resource on edge server s_j at time t which can be utilized by hosted services. Each server s_j has a coverage radius R_j within which the server can exclusively cater to service requests from the users.

We denote by W_l the demand vector $\langle CPU, RAM, storage, bandwidth\rangle$ of QoS level l, denoted as $\langle w_l^1, w_l^2, w_l^3, w_l^4\rangle$ in that order. For user u_i, the preferred QoS level is denoted as H_i^t, and the threshold L_i^t for the lowest QoS level tolerable. An user allocation policy can assign the user at *any* QoS level between its lowest L_i^t and highest H_i^t QoS preference at time t.

Notations	Descriptions
S^t	Set of all active servers at time t, $s_j \in S^t$
U^t	Set of all active users at time t, $u_i \in U^t$
C_j^t	Capacity vector $\langle CPU, RAM, storage, bandwidth\rangle$ of s_j at t denoted as $C_j^t = \langle \left(c_j^1\right)^t, \left(c_j^2\right)^t, \left(c_j^3\right)^t, \left(c_j^4\right)^t \rangle$ in that order
q	Number of QoS levels
W_l	Demand vector $\langle CPU, RAM, storage, bandwidth\rangle$ of QoS level l denoted as $\langle w_l^1, w_l^2, w_l^3, w_l^4\rangle$ in that order
E_{il}^t	QoE value for user u_i at QoS level l at time t
H_i^t	Preferred QoS level of user u_i at time t
L_i^t	Threshold QoS level of user u_i at time t
q_i^t	QoS level assigned to u_i at time t
d_{ij}^t	Distance between user u_i and server s_j at time t
R_j	Signal range / Radius of server s_j
Δ_{ij}^t	Latency experienced by u_i allocated to s_j at t
δ	Latency Upper Bound

Table 3.3: List of Notations

The aim of an allocation policy is to serve the maximum number of users at their preferred QoS level, thereby, maximizing the overall QoE of all users. In addition, it needs to ensure the resource utilization by services does not exceed server capacity due to user-server binding. Moreover, the allocation policy should not allocate an user to an edge server, if the user is not within the coverage radius of the server, thereby respecting the coverage constraint induced by the relative distance between users and servers.

Users who do not get allocated to the servers due to a shortage of resources need to wait till the resources on the edge server are made available. We assume a set of q QoS levels. Let E_{il}^t denote the QoE value for u_i at QoS level l, q_i^t the QoS level assigned to u_i at time t, d_{ij}^t the distance between u_i and server s_j, Δ_{ij}^t the latency experienced by u_i allocated to s_j at t.

We compute latency Δ_{ij}^t as a function of q_i^t and d_{ij}^t. The latency experienced in any user-server allocation has to honour a maximum limit denoted by δ. We formulate an Integer Linear Program (ILP) for the problem below.

Objective:

$$Maximize: \sum_{t \in T} \sum_{i=1}^{|U^t|} \sum_{j=1}^{|S^t|} \sum_{l=L_i^t}^{H_i^t} x_{ijl}^t \times E_{il}^t \tag{3.1}$$

where,

$$x_{ijl}^t = \begin{cases} 1, & \text{If user } u_i \text{ is allocated to server } s_j \text{ at QoS level } l \text{ at time } t \\ 0, & \text{Otherwise} \end{cases}$$

Subject To:

1. Coverage Constraint:

$$d_{ij}^t \leq R_j^t \tag{3.2}$$

2. Capacity Constraint:

$$\sum_{i=1}^{|U^t|} \sum_{l=L_i^t}^{H_i^t} w_l^k \times x_{ijl}^t \leq \left(c_j^k\right)^t \quad : \forall t \in T,\ \forall j \in \left\{1, \ldots |S^t|\right\},\ \forall k \in \{1, \ldots 4\} \tag{3.3}$$

3. Latency Constraint:

$$\sum_{j=1}^{|S^t|} \sum_{l=L_i^t}^{H_i^t} \Delta_{ij}^t \ \times x_{ijl}^t \leq \delta \quad : \quad \forall t \in T,\ \forall i \in \left\{1, \ldots |U^t|\right\} \tag{3.4}$$

4. User-Server Mapping:

$$\sum_{j=1}^{|S^t|} \sum_{l=L_i^t}^{H_i^t} x_{ijl}^t \leq 1 \quad : \quad \forall t \in T,\ \forall i \in \left\{1, \ldots |U^t|\right\} \tag{3.5}$$

5. Integer Constraint:

$$x_{ijl}^t \in \{0, 1\} \quad : \forall t \in T,\ \forall i \in \left\{1, ..|U^t|\right\},\ \forall j \in \left\{1, ..|S^t|\right\},\ \forall l \in \left\{L_i^t..H_i^t\right\} \tag{3.6}$$

The aim of the objective function is to maximize the overall QoE of users over the set of time slots $t \in T$ where T is the total time period for evaluation. The indicator variable x_{ijl}^t at any time instant t denotes all possible user-server-QoS preferences. The summation on the indicator variable encodes all personal preferences and thresholds without explicitly specifying the minimum required QoS level.

The coverage constraint in Equation 3.2 ensures that at any time instant t, a user u_i can be allocated to s_j if the user is within radius R_j. To allocate u_i to s_j at a QoS level l, the resource requirement at s_j is

denoted by W_l. The total resources allocated must honour the capacity constraint of each server. The edge server capacity constraint expressed in Equation 3.3 ensures that the combined requirements of users allocated to a server does not exceed the server's total capacity for each dimension CPU, RAM, storage and bandwidth of the resource vector. Equation 3.4 expresses the latency threshold constraint which ensures users are allocated to servers such that a predefined latency threshold is honoured. Equation 3.5 is used to express that a single service can only be allocated to a single server at a QoS level at any t. Equation 3.6 specifies that x^t_{ijl} variables are Boolean indicator variables denoting service requests from users, the respective server to which the requests are allocated and required QoS values.

As observed in [31], QoS is non-linearly correlated with the QoE for any service. We formulate the QoS-QoE correlation using a similar logistic function (Eq. (3.7)) as in [31]. However, we include an additional scaling to incorporate the QoS level preference and lowest threshold as specified by the users. The QoE E^t_{il} experienced by u_i at time t for level l is expressed as:

$$E^i_l = \frac{E_{max}}{1 + \exp\left\{-\alpha\left(\gamma^t_{il} - \beta^t_i\right)\right\}} \tag{3.7}$$

The scaling in the logistic function helps to assign the lowest QoE value to the lowest QoS level, similarly, the highest QoE value to the highest QoS level. For a user u_i, E^t_{il} depends on the QoS level W^t_l, his QoS preference H^t_i and the threshold level L^t_i at time t. Here, $\gamma^t_{il} = \frac{\sum_{k=1}^{4} w^k_l}{4}$ is the mean computational demand of QoS level W_l of user u_i at time t; $\beta^t_i = \frac{\gamma^t_{iH^t_i} - \gamma^t_{iL^t_i}}{2}$ is the mean QoE value of user u_i at t. The value E_{max} represents the maximum value of QoE and α is the growth factor of the logistics function.

The allocation solution generated by the presented ILP formulation gives an optimal user-server-QoS binding policy, honouring QoS preferences of each user, the latency upper bound and coverage constraints. If the ILP solver returns infeasible, we conclude the user settings cannot be allocated to their proximate edge servers, given the constraints. The proposed ILP is event-driven i.e. to consider user mobility and preference change, we re-evaluate the ILP to obtain a new solution. The events to trigger the re-evaluation of ILP are following:

- Any user changes the QoS specification
- Users or edge-servers become inactive
- Users move in and out of the service zone of servers, and
- New service requests are placed.

However, due to the exponential complexity of the problem [31], re-evaluating the ILP frequently turns out to be a non-scalable strategy, as demonstrated in our experimental results presented in Section 3.4. To address this, we design a scalable heuristic to cater to real-world dynamic scenarios.

3.3 Proposed Heuristic Solution

In this section, we present the design of an efficient polynomial-time heuristic that generates near-optimal solutions. Algorithm 3.1 outlines our method where we use a Red-Black (RB) Tree [20] as an

indexing data structure. The algorithm maintains a Red-Black Tree for each edge server and uses a metric defined as i-factor for each user in its service zone as an index. This heuristic is used in place of the ILP and is run whenever any of the events mentioned earlier occur, necessitating a reevaluation of the allocation. However, this being a polynomial-time algorithm, is lightweight and can be executed more efficiently than the ILP. Algorithm 3.1 summarizes the main steps in our approach. It includes the following steps:

- Lines 1-7 presents the task of dividing the new users into two classes, single-server class (S-class) and multi-server class (M-class). The users within the range of only one edge-server are clustered into S-class and the users within the range of more than one edge-server are put into the M-class. For example, in Figure 3.1, the users u_1, u_2, u_3, u_5 and u_6 are within the range of only one server i.e. E_1 and are hence clustered into S-class. However, u_4 can access both E_1 and E_2, hence gets clustered into the M-class. This categorization is done once for all users at the start and adjusted at every time slot only if there is a change in user locations, new users join in, or existing users leave.

- The users in both S-class and M-class are allocated an initial QoS level at their minimum threshold specified. It may be noted that if any user cannot be assigned in his least preferred QoS level, then it is impossible to assign any further arrangements. Referring to the scenario in Section 3.1, u_1, u_2, u_3 u_4, and u_5 are initially assigned with QoS level of W_1, W_1, W_2, W_1 and W_2 respectively. The increment factor (i-factor), discussed later in this section, is computed for all the S-class and M-class users. The i-factor is determined by the user's QoS preference and presently assigned QoS level ($plevel$). S-class is considered before the M-class since S-class users are bound to a single edge server for determining the allocation. Each user is assigned to the edge server according to his i-factor. Users with a low i-factor receive a higher preference for an edge server during the assignment. For M-class users, the allocation policy tries to assign a user to his nearest local server with the required remaining computation resource, with a motivation to serve him with a better latency experience. Line 8 sorts the users according to their i-factor. Lines 9-17 compute the initial assignment and update the Red-Black Tree with i-factor as key for each server.

- Our heuristic then attempts to enhance the QoS level of each user (upper bounded by their respective preference levels) and re-evaluates the i-factor after incrementing the QoS level. This process continues till all users receive their QoS preference levels or the server exhausts its available resources. We move on to examine the following server in the vicinity of the user from where he can be served. Lines 18-19 perform this update.

- For servers that have exhausted their resources, users from M-class may be migrated to the other nearby servers having free resources. Lines 20-21 execute this migration. Once users have been migrated to nearby servers, the QoS levels have to be re-evaluated. QoS upgrade is therefore re-performed after migration in Lines 22-23.

The heuristic algorithm selects the user with the smallest i-factor and increments the QoS level of that user. It then proceeds to update the Red-Black Tree with the re-computed i-factor. Considering our example, at $t = 0$, on enhancement of QoS levels, the users $u_1 \dots u_5$ are allotted W_1, W_2, W_3, W_2 and W_3 respectively. We keep track of the left-most child for updating the QoS level and reinsert the user after evaluating the $ifactor$ as per the new QoS assignment. Algorithm 2 runs concurrently for each edge server. It selects the lowest i-factor from the Red-Black Tree and increases the QoS level of the user by one. It then updates the i-factor of the user before re-inserting the updated i-factor into the Red-Black Tree.

Algorithm 3.1: Edge User Allocation with User QoS Preferences

Input : $U \leftarrow$ Set of Users, $S \leftarrow$ Set of Servers
Output: User-Server Allocation and respective QoS Levels

foreach $u \in U$ **do**
  $plevel \leftarrow$ least QoS preference of user u
  $ifactor \leftarrow$ compute $ifactor$ of user u
  if *u is in range of single server* **then**
    $U_{sclass} \leftarrow U_{sclass} \cup u$ ▷ Assign user to S-class
  end
  else
    $U_{mclass} \leftarrow U_{mclass} \cup u$ ▷ Assign user to M-class
  end
end
Sort (U_{sclass}, U_{mclass} users according to $ifactor$)
foreach $u \in U_{sclass} \cup U_{mclass}$ **do**
  $s_{list} \leftarrow$ list of servers accessible to u sorted in ascending order of distance
  foreach $s_u \in s_{list}$ **do**
    if *s_u has remaining capacity to accommodate user u* **then**
      if *Red-Black Tree for s_u is not initialized* **then**
        initialize Red-Black Tree for server corresponding to s_u
        $S_{active} \leftarrow S_{active} \cup$ server corresponding to s_u
      end
      insert user into Red-Black Tree of server s_u
      decrease resources available at server corresponding to s_u
    end
  end
end
foreach $s \in S_{active}$ **do**
  $QosIncrement(s)$
end
foreach $s \in S_{active}$ **do**
  $S_{modified} \leftarrow MigrateUsers(s)$
end
foreach $s \in S_{modified}$ **do**
  $QosIncrement(s)$
end

Algorithm 3.2: QosIncrement(s)

Input : $U \leftarrow$ Users, $S \leftarrow$ Servers, $S_{object} \leftarrow$ Server Red-Black Trees
while *$s \in S_{object}$ has resources OR Red-Black Tree is not empty* **do**
  $u \leftarrow$ pick the left most child from Red-Black Tree
  if *user u has not reached its maximum QoS preference* **then**
    $plevel = plevel + 1$
    $ifactor \leftarrow$ compute new ifactor according to new QoS level
    insert u into Red-Black tree using new ifactor
  end
end

Algorithm 3.3: MigrateUsers(s)

Input : $U \leftarrow$ Users, $S \leftarrow$ Servers, $S_{object} \leftarrow$ Server Red-Black Trees
Output: Returns list of servers from/to which users were migrated $S_{modified}$

```
1 foreach s_object ∈ S_object do
2     foreach mclass user u_m assigned to s do
3         migrate u_m to nearby server with requisite residual resource capacity
4         s_to, s_from ← servers from which and to which u_m was migrated respectively
5         S_modified ← S_modified ∪ s_to ∪ s_from
6     end
7 end
```

Computation of i-factor : The i-factor helps to determine which user causes more alterations to QoE values if the QoS level is increased. The factors which affect the QoE value are the user's preference values (low and high) and the present QoS level ($level$) assigned to the user. The variable E_{max} and function QoE are determined using Equation 3.7 discussed in the previous section. Users with lower i-factor values are given higher preferences when the QoS values allocated to them are upgraded. Equation 3.8 determines the i-factor of a certain user u_i having level preference and threshold of H_i^t and L_i^t respectively with presently assigned QoS level of l at time t. The QoE function E_i^t, E_{max} and α are from Equation 3.7 discussed earlier. The numerator affects the i-factor by scaling the QoE value according to the present QoS level, i.e., it assigns a higher i-factor as users reach their preferred QoS levels. The denominator demarcates the difference between H_i^t and L_i^t, the higher the difference, the lower is i-factor.

$$ifactor = \frac{E_{max} \times (E_i^t + l)}{\alpha \times max(H_i^t - L_i^t, 1)} \tag{3.8}$$

Migrating Users for Improving QoE: Once all the Red-Black trees corresponding to all edge servers have been updated, Algorithm 3.3 determines the list of users that can be migrated for those servers which have exhausted their resource capacities and hence, no further QoS up-gradation for users are possible. Upon successful migration, Algorithm 3.2 is re-initiated for possible QoS up-gradation.

Running time analysis of Heuristic: Let the number of users be n and the number of servers be m. The classification of users into S_{class} and M_{class} in line 1 takes $O(n)$; sorting of users takes $O(nlogn)$; insertion of S_{class} users and M_{class} users into RB Tree takes $O(nlogn)$ and $O(nmlogn)$; the updation of QoS requirements incurs $O(logn)$; the migration algorithm runs in $O(nmlogn)$. Hence, the heuristic algorithm takes total of $O(mnlogn)$.

3.4 Experiments and Analysis of Results

All experiments for this chapter are conducted on a machine with an Intel Core i5-8250U processor and 8GB RAM. The Python Mixed-Integer-Programming library [3] is used to solve the ILP model discussed in Section 3.2. The results from our heuristic are compared with the baseline ILP formulated in Section 3.2, the optimal algorithm presented in [31] and the dynamic mobility aware policy in [47].

3.4.1 Experimental Setup

We use the EUA data-set for the location of edge servers and edge users. The data set includes location data of base stations and users within the Melbourne Central Business District area. The coverage area of edge servers are set randomly to values between 200-400 meters radius. To simulate various attributes of users over time, we arbitrarily select several users and include the following:

- To simulate the movement of users we randomly assign 20% users with $0m/s$ speed to represent static users, 30% users with random speed between $1 - 2m/s$, the average walking speed of humans, and the remaining 50% users with speed between $10 - 20m/s$, the average vehicle speed in the city.
- We randomly assign an initial direction between 0° to 360° which then follows the random way-point mobility model [47].
- We randomly assign the user high and low QoS preference levels.

We used the real world PlanetLab and Seattle latency data-set [80] to generate latencies. The PlanetLab and Seattle latency data-set comprise latencies from across the world. As it does not adequately represent the latencies in a realistic MEC environment, we cluster the data-set into 400 clusters considering devices that are in proximity of each other. We randomly pick a cluster and the representative latency is assigned according to our latency measure derived based on the distance and QoS level similar to [67]. The latency measure is essentially the product of distance and QoS level, which is scaled down according to the number of clusters.

We consider a discrete-time slotted model with each slot of $25s$ in which the users move and configure their QoS preferences dynamically. Hence, each time slot is followed by modification of some user's location. To simulate dynamic QoS preference changes, we assign randomly a new QOS preference level to 20% of users. The number of discrete-time slots is kept at 20 for each experiment.

To consider various sizes of user and server populations, we consider the two groups of data summarized in Table 3.4. In Group 1, we vary the number of users from 50 users to 250 users at intervals of 50, while keeping the number of servers to 50 and the server resources at 100% of the cumulative resource requirement of all users at the highest QoS level, distributed uniformly over all servers. Similarly, in Group 2 we vary the number of servers from 10 to 100 at intervals of 10, while keeping the number of users fixed at 500 and server resources to 100% of the cumulative resource requirement of all users at the highest QoS level, distributed uniformly over all the edge servers. Each experiment is averaged over 50 runs. For the QoE model, we set $E_{max} = 5$, $\alpha = 1.5$.

Group Number	Number of users	Number of servers
1	$50, 100 \ldots 250$	50
2	500	$10, 20 \ldots 100$

Table 3.4: Experiment settings for number of users and servers

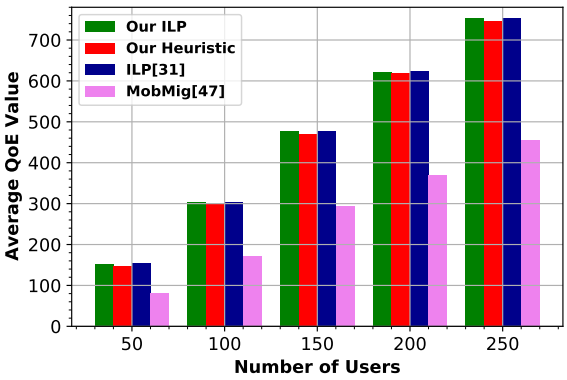
We compare the results of our ILP, our heuristic, the static ILP proposed in [31] and MobMig [47], a Mobility-Aware dynamic allocation policy. We consider the ILP in [31] by running it in each discrete time step since it is a static formulation. We use MobMig by setting the QoS level at the highest possible since MobMig does not support dynamic QoS changes. For comparison, we study the following metrics:

a. Average QoE achieved per time slot.

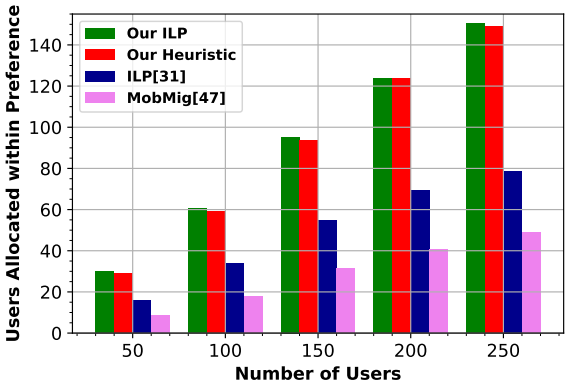
b. Average number of users allocated within their QoS preference per time slot.

c. Average latency experienced by users.

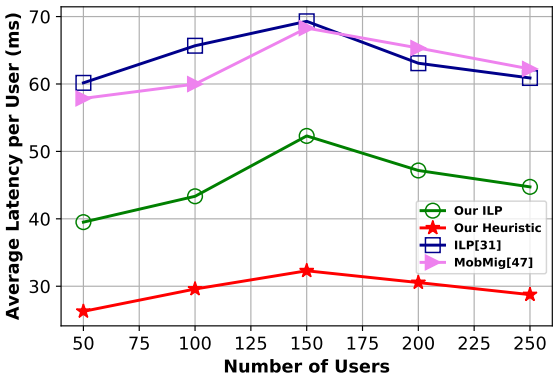
d. Average execution time (CPU time) for evaluation of algorithms.



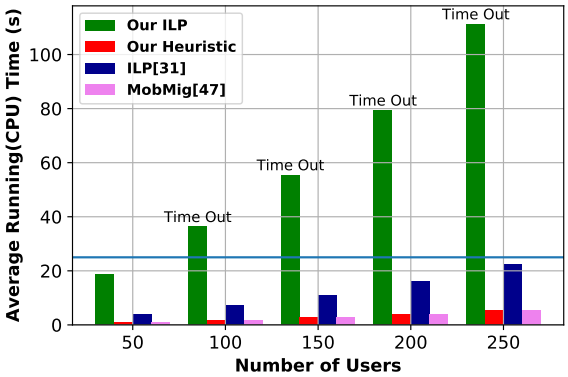
(a) Average QoE achieved per time slot



(b) Average number of users allocated within their QoS preference per time slot



(c) Average latency experienced by users



(d) Average execution time (CPU time) for evaluation of algorithms

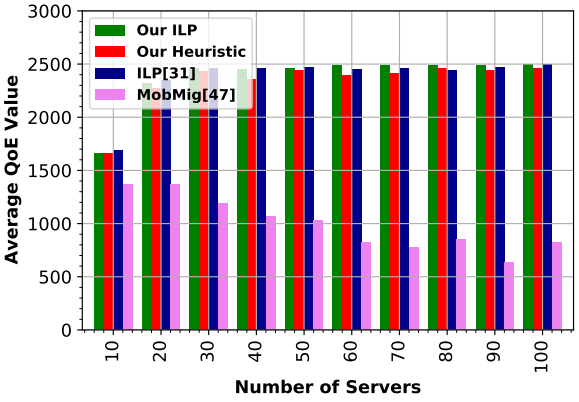
Figure 3.2: Group 1 Experiment: We vary the number of users and fix the number of edge servers to 50 to obtain the results for various metrics

3.4.2 Results and Discussion

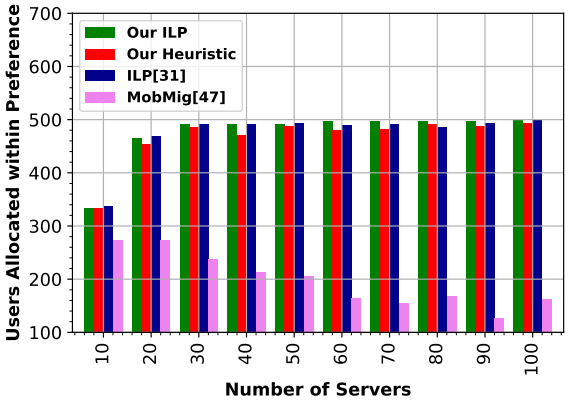
Figures 3.2 and 3.3 depict the experimental results for Group 1 and Group 2 respectively as in Table 3.4 across four different metrics discussed earlier.

Discussion on effectiveness:

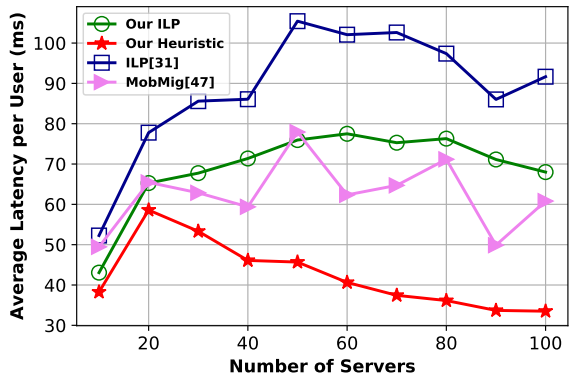
The results demonstrate the effectiveness of the heuristic in being able to generate near-optimal solutions for both average QoE and the average number of users allocated. The average QoE and number of users allocated by the heuristic is nearly comparable with the results from the optimal ILP which can be inferred from Figure 3.2a and 3.2b respectively. A similar trend is observed for the results of Group 2 experiments. Migration of users from overloaded servers to under-loaded servers has a key



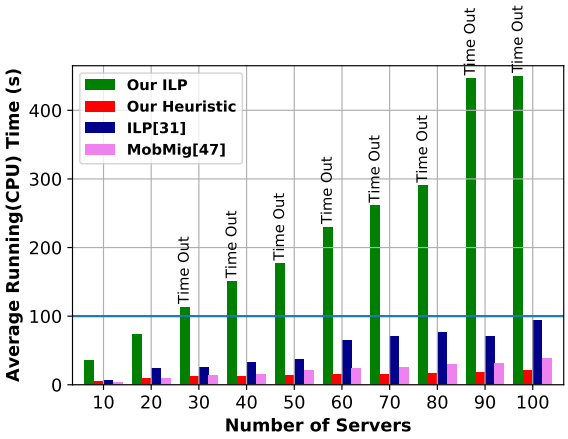
(a) Average QoE achieved per time slot



(b) Average number of users allocated within their QoS preference per time slot



(c) Average latency experienced by users



(d) Average execution time (CPU time) for evaluation of algorithms

Figure 3.3: Group 2 Experiment: We vary the number of edge servers and fix the number of users to 500 to obtain the results for various metrics

role in achieving near-optimal QoE. MobMig [47], being unaware of users' QoS preferences allocates users at the highest available QoS level. Consequently, the allocation policy fails to accommodate a substantial fraction of the users as can be inferred from Figures 3.2b and 3.3b. However, the ILP [31], which seeks to optimize overall QoE, generates near equivalent QoE and number of allocated users as compared to our ILP and heuristic. In Figure 3.3, average QoE value and number of users allocated do not follow an increasing trend unlike in Figure 3.2 since, in Group 2, we keep the total number of users fixed while varying the number of servers.

Discussion on latency:

The average latency per user is depicted in Figures 3.2c and 3.3c. As can be inferred from the figure, both our optimal and heuristic policies significantly outperform MobMig and the ILP in [31] in terms of average latency incurred by the users. This is because our preference aware policies provide the flexibility to dynamically adapt to QoS values depending on user-QoS preference levels and hence conserve resources both at the server end and at the user end. Additionally, at the user-end, adapting to varying QoS levels prevent higher communication data transfer latencies. As such, our heuristic, which initially assigns the lowest assignable QoS value to users, while progressively upgrading the QoS values depending on resource availability, results in a considerably lower average latency owing to lower communication overhead.

Discussion on time:

Figures 3.2d and 3.3d depict the effectiveness of our algorithm in a mobility-driven dynamic scenario where the heuristic takes a fraction of the running time of our ILP. Our heuristic requires lower running times as compared to MobMig and the ILP in [31] while at the same time being QoS-preference aware with lower latency overheads.

3.5 Conclusion

In this chapter, we propose a novel user-centric approach to obtain an user server allocation honoring user QoS preferences in a dynamic MEC environment. We formulate an optimal ILP for a user-centric approach. Since the ILP solution does not scale to a real-world scenario, we propose a near-optimal heuristic. We demonstrate the effectiveness of our proposed algorithm using a real-world EUA dataset. The results of our proposed algorithm are inspiring for real-world deployment.

The user-centric policy proposed in this chapter relies on user QoS preferences as input. However, obtaining these preferences each time the user requests a service from an edge server is tedious. In the next chapter, we put forward a mechanism to address this limitation with stochastic estimates of service resource consumption.

Chapter 4

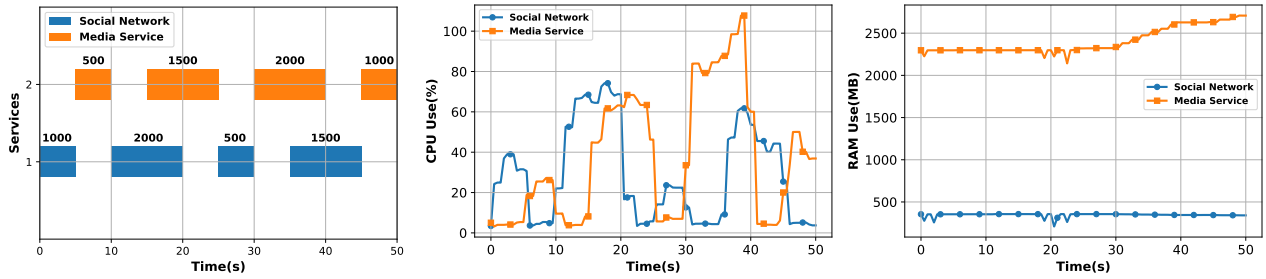
Service Allocation and Placement with Workload Fluctuations

4.1 Introduction

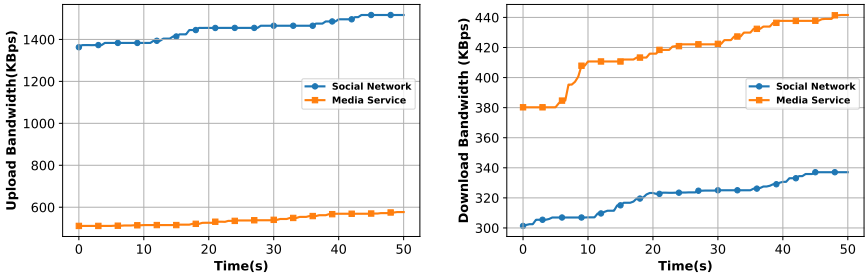
The QoS level of a service directly affects the resource utilization on the edge server. For example, the graphics of a mobile game rendered at the most superior quality consumes more resources than their lower quality counterparts. In the previous chapter, we used the QoS level preferences of users to determine the allocation policy. However, that approach requires users to inform their QoS level preferences. The most formidable hurdle in that approach is to collect the information on the QoS level preferences of the user. How should we obtain that? A straightforward solution could be to integrate an option for the QoS preference levels within all applications. Users can manually set a QoS level as per their preferences. But this simplistic manual approach is not at all convenient for real-time applications like mobile games. The user will have to pause the game and manually switch the QoS level, which is a daunting obstacle from the user's perspective.

Additionally, with the paradigm shift towards a microservice architecture where monolithic services are broken down into small microservices with intercommunications between them represented as DAGs [23], resource utilization may vary [24], depending on which services are required to satisfy a service invocation. Consequently, service requests may face high latencies if the service allocation and placement policies are not well-adapted to handle load variation. In this chapter, we exploit the influence of a QoS level on service resource utilization on an edge server to obtain an allocation policy that considers the workload fluctuations on that edge server.

The main contribution in this chapter is a joint optimization approach for user request allocation and container placement considering runtime load variations of service execution arising due to heterogeneous service invocations and their consequent resource demands, and edge server capacity constraints. Evidently, the runtime workload at each server is dependent on the cumulative demand of the resource footprint of the service containers to be hosted there. The primary challenge is in deriving an estimate of the amount of resources that may need to be allocated to each container at runtime for execution, which is often difficult to learn or predict based on past usage records, due to the dynamic nature of the services, the invocation patterns and the resulting execution variations. The size estimate should ideally match the workload of a service container in execution on the user-requested service inputs. Such sizing is often done conservatively (i.e. overestimation) and often in isolation i.e. without any consideration of the workload of the co-located user requests. Such over-provisioning of resources for service requests based on maximum requirement may lead to resource under-utilization during



(a) Service Invocation Gantt Chart (b) CPU Utilization (c) Memory Utilization



(d) Uplink Bandwidth (e) Downlink Bandwidth

Figure 4.1: Resource Usage Fluctuations corresponding to the 50 second interval as in Figure 4.1a

execution, and thus deny other user requests from being onboarded due to lack of server capacity to host the corresponding service containers. At the same time, under-provisioning may lead to unsatisfactory execution performance due to demand overflows. In reality, the peak and valley in resource demand for one service request do not necessarily coincide with other requests. Our proposal here is based on the fact that a tighter allocation or packing can be achieved leading to better utilization of resources, by exploiting stochastic models for sizing and multiplexing them statistically for estimating aggregate runtime demands of the containers co-allocated on each edge server at runtime.

To demonstrate fluctuations in the workload under varying conditions, we consider a simplified Virtual Machine with 8 vCPUs and 5 GB of RAM representing a MEC server where we deploy services and simulate users making service requests to the MEC server. We deploy the services Social Network and Media Service on this Virtual Machine from the DeathStar Microservice benchmark [23]. We then simulate users invoking service requests to both these applications over 50 seconds. The service invocation pattern simulated is outlined in Figure 4.1a. Blue and Orange blocks represent the duration during which the corresponding service is invoked while the number of users invoking the services in that particular duration is indicated by the number above the block. For example, 1000 users invoke the Social Network service in the $0-5$ seconds interval while 500 users invoke the Media Service in the $5-10$ seconds interval. Additionally, in the interval from $15-20$ seconds, the Social Network and Media Service are both invoked by 2000 and 1500 users respectively. Figure 4.1 depicts the resource usage patterns during this 50 seconds interval. As can be inferred from the figure, depending on the number of user requests and how many services are active simultaneously, the resource usage varies over time. Such a scenario motivates the need to re-analyze traditional policies which make allocations and placement decisions agnostic of such variations.

In this chapter, we formulate a stochastic optimization model for the joint service allocation and placement problem and solve a stochastic programming formulation to generate optimal solutions through determinization. We model the size of a service request in terms of the resource elements and treat these as random variables, thereby effectively representing their stochastic nature. We consider the following four types of resources as in [49]: a) memory capacity needed to host the data associated with services, this includes both the requirements of the service container itself and the

data required at runtime, b) CPU computation capacity (measured as the number of virtual CPUs or otherwise) needed to execute the requested services, c) uplink bandwidth capacity, and d) downlink bandwidth capacity. Evidently, each of the parameters above is susceptible to run time variations and need to be examined through a stochastic lens for better allocation and placement. We consider a static offline optimization model in this work, as has been done in several MEC studies [30, 31, 48]. While dynamic approaches are better suited to adapt to runtime variations, their main limitation is the amount of non-trivial computation that is needed at runtime for executing the optimization model. Static stochastic approaches that incorporate variations are often good approximations to their dynamic counterparts and widely used in practice. Such stochastic models have been demonstrated to effectively capture dynamic behaviour in the context of resource allocation in Cloud Computing [41] and runtime service composition in Web Services [12, 13]. We use stochastic formulations to model MEC service placement and allocation here.

We present experiments on the EUA dataset, a real-world MEC benchmark. We compare our results with state-of-the-art approaches that do not consider workload fluctuations to show that our framework fares better in terms of lesser runtime overflows and more user onboarding on the edge servers.

The rest of this chapter is organized as follows. Section 4.2 motivates the problem context with an example. Section 4.3 proposes the stochastic ILP model and solution. Section 4.4 presents the experimental results. Section 4.5 concludes the chapter.

4.2 A Motivating Example

In this section, we present a motivating example to explain the context addressed in this chapter. Consider a scenario with 4 users u_1, u_2, u_3 and u_4 in an MEC environment under the coverage area of one edge server e. As mentioned earlier, in the event that a user's service request cannot be allocated to any edge server, it is allocated to a distant cloud server E_c. In this example, for the sake of simplicity, we illustrate using only the memory required. We later generalize this to the CPU, uplink and downlink bandwidth requirements. For each user service request invocation, Table 4.1 shows the minimum, the average and maximum memory footprints in GB required for service containers for services s_1, s_2, s_3, s_4. Consider the scenario that u_1 invokes s_1, u_2 invokes s_2, u_3 invokes s_3 and u_4 invokes s_4. Assume e has a memory capacity of 2GB while the cloud has infinite memory i.e. any number of users can be allocated to the cloud. However, connecting to a distant cloud increases the latency perceived by the user. The goal is to allocate as many user requests as possible on the edge servers, considering latency implications.

Services	Memory Needed		
	Minimum	Average	Maximum
Online Games (s_1)	0.5	0.6	1.10
Maps (s_2)	0.3	0.6	0.88
Face Recognition (s_3)	0.6	0.8	1.12
File Compression (s_4)	0.4	0.6	1.11

Table 4.1: Service Request Record

Since we have a single edge server, the problem is now to decide which containers to host at e, considering the corresponding memory requirements. Consider an allocation strategy that uses the minimum memory requirement values for allocating RAM to the service containers for s_1, s_2, s_3 and s_4. In our example scenario, this will mean placing all 4 containers and allocating all four users at e, provisioning a total of 1.8GB (0.5 + 0.3 + 0.6 + 0.4), which is less than the total RAM capacity of 2

GB. However, this may lead to runtime overflows. Consider the worst case scenario wherein all users $u_1 \dots u_4$ actually present the maximum memory requirement values instead at runtime, resulting in a total need of 4.13GB (1.10 + 0.8 + 1.12 + 1.11). This exceeds the resource capacity of e, leading to latency hits since each user may need to wait until the required memory resources have been freed.

To avoid runtime overflow scenarios, an allocation strategy may prefer an overtly conservative approach by using the maximum requirements of the resource containers for s_i (Column 4). In our example scenario, this will onboard any of the pairs $\langle u_1, u_2 \rangle$, $\langle u_2, u_3 \rangle$, $\langle u_2, u_4 \rangle$ on e and the rest on the cloud server e_c. Such an approach, however, may lead to under-utilization of resources of e in the event that the actual user service invocation requests place much lower memory demands at runtime. An allocation strategy which is neither overtly aggressive nor overtly conservative in allocating resources, may allocate based on the average requirement values assuming these are sufficiently representative of the resource utilization of the service containers. In our example, such an approach assigns u_1, u_2 and u_4 to e utilizing a total of 1.8GB (0.6 + 0.6 + 0.6) on e. However, such an approach can lead to overflows / under-utilization at e at runtime as well.

The main limitation of the above static approaches is due to the assumption that the resource requirements for the future service invocations are either known apriori or follow a similar trend as in the past usage instances. However, the past usage values are not always sufficiently representative of the future resource needs. This is due to the fact that the memory requirements of the service containers hosted at the edge servers are quite different, when invoked on different inputs and invocation patterns. Thus, for the same user, for the same service invocation, the runtime memory requirements may be quite different from the past values, considering the variation induced by the service inputs and the consequent processing needs. On a similar note, the CPU cycles needed to process a user requested service container may be different as well, depending on the nature of the service being invoked, which may vary over time for the same user in the same geographical area. At the same time, the downlink bandwidth needed to download service invocation inputs (e.g. image feeds from vehicles) for a user service request, and the uplink bandwidth needed to send response packets to the requesting entity may vary at runtime as well.

To address these issues, we introduce a generic stochastic constraint formulation for the user server allocation and service placement problem considering the parameters memory, CPU, bandwidth as random variables. With this model, we propose a formulation for the joint allocation and placement problem in two different situations, namely, (a) when the distribution of each parameter is known, and (b) when the distributions are unknown. In both the cases, we formulate stochastic constraints and present solutions that satisfy the constraints probabilistically, as is usually done in stochastic programming. We discuss our approach in the following.

4.3 Problem Formulation

We have the following in our problem context:

- A set of edge services $S = \{s_1, s_2, \dots, s_p\}$
- A set of edge users $U = \{u_1, u_2, \dots, u_n\}$
- A set of service requests $R \subseteq S$ from users in U for accessing services from S, with $R = \{r_1, r_2, \dots, r_k\}$. Each request r_i is owned by an user $u(r_i) \in U$ and refers to some service in S.
- A set of edge servers $E = \{e_1, e_2, \dots, e_m\}$.

- For each edge server, we have a memory capacity Q_e, a CPU compute capacity C_e, an uplink bandwidth capacity $B_e^{\uparrow}$ and a downlink bandwidth capacity $B_e^{\downarrow}$.
- For each request $r_i \in R$, we have a set of parameters:
  - Memory requirement, a random variable s_{r_i} following some probability distribution function (either known or unknown) with mean $\mu_s^{r_i}$ and variance $(\sigma_s^{r_i})^2$
  - CPU requirement, a random variable (c_{r_i}) following some probability distribution function (either known or unknown) with mean $\mu_{r_i}^c$ and variance $(\sigma_{r_i}^c)^2$
  - Up-link bandwidth, a random variable $(b_{r_i}^{\uparrow})$ following some probability distribution function (either known or unknown) with mean $\mu_{r_i}^{b^{\uparrow}}$ and variance $(\sigma_{r_i}^{b^{\uparrow}})^2$
  - Down-link bandwidth, a random variable $(b_{r_i}^{\downarrow})$ as above with mean $\mu_{r_i}^{b^{\downarrow}}$ and variance $(\sigma_{r_i}^{b^{\downarrow}})^2$

Further, as part of our stochastic formulation, we have a bound on each parameter $Q_e, C_e, B_e^{\uparrow}$ and $B_e^{\downarrow}$ to be satisfied with probability $\geq 1 - \alpha$, where α is the overflow probability. The essential idea is to place containers and onboard users on the edge servers, while satisfying capacity constraints on the server parameters probabilistically. To develop the stochastic model, we first discuss a simple integer programming model below based on the one in [49], considering all users are covered by all edge servers for simplicity. We later dispense with this requirement when we present our formulation.

Let E_c represent the cloud server. Let $E_u \subseteq E$ denote the set of edge servers covering user $u \in U$. Note that a single user can be under the coverage area of multiple edge servers. The set $S_e \subseteq S$ denotes the set of services deployed at the server $e \in E$. Each edge server can host a number of services from S having different resource requirements. Request from an user $u \in U$ under the coverage of an edge server $e \in E$ can be allocated provided that the service container can be hosted at the edge server with the required computation and bandwidth resources. The cloud server E_c hosts all available services. In the case that a user is not allocated to any edge server due to resource constraints, we allocate the user to E_c.

4.3.1 A simple optimization model

We first present a standard Integer Linear Programming (ILP) formulation for the allocation and placement problem. The ILP formulation below attempts to minimize the number of users sent to the cloud server. We use a binary (0/1) decision variable v_{ij} for the ILP formulation, where $v_{ij} = 1$ denotes service request r_i is onboarded on some edge server $e_j \in E$. Let $R_e \subseteq R$ denote the set of requests allocated to edge server $e \in E$. Thus, for each request r_k in R_e, we have $v_{ke} = 1$.

Objective:

$$Maximise : \sum_{r_i \in R} v_{ij} \tag{4.1}$$

Subject To:

$$R_e \cap R_{e'} = \phi, \quad where\ e, e' \in E\ and\ e \neq e' \tag{4.2}$$

$$\sum_{r_k \in R_e} c_{r_k} \leq C_e, \quad \forall e \in E \tag{4.3}$$

$$\sum_{r_k \in R_e} s_{r_k} \leq Q_e, \quad \forall e \in E \tag{4.4}$$

$$\sum_{r_k \in R_e} b_{r_k}^{\uparrow} \leq B_e^{\uparrow}, \quad \forall e \in E \tag{4.5}$$

$$\sum_{r_k \in R_e} b_{r_k}^{\downarrow} \leq B_e^{\downarrow}, \quad \forall e \in E \tag{4.6}$$

Equation 4.1 is the optimization objective. Any feasible allocation needs to restrict each user to be allocated to only 1 edge server, expressed by Equation 4.2. Equations 4.3, 4.4, 4.5 and 4.6 ensure that the combined requirements of the requests allocated to any edge server satisfy memory, CPU, uplink, downlink capacities.

4.3.2 Stochastic Optimization Model

The above constraints involve random variables c_{r_k}, s_{r_k}, $b_{r_k}^{\uparrow}$, $b_{r_k}^{\downarrow}$ and cannot be directly solved by ILP solvers. To deal with the randomness of resource elements, we formulate probabilistic capacity constraints with bounding values to express the requirement that the capacity constraints at each edge server for each of the four dimensions have to be satisfied with a certain probability. This is in sharp contrast to allocation and placement methods that treat these as constants, and formulate optimization models to satisfy cumulative resource bounds on the same for each edge server. To this end, we define an *overflow probability* α between 0 and 1. Consequently, the probability $1 - \alpha$ represents the probability of the event where the allocation strategy of users to the edge server does not overflow, hence total resource utilization by the services are within the edge server resource capacity. Hence, the probabilistic version of resource constraints are below:

$$P\left[\sum_{r_k \in R_e} c_{r_k} \leq C_e\right] \geq 1 - \alpha, \quad \forall e \in E \tag{4.7}$$

$$P\left[\sum_{r_k \in R_e} s_{r_k} \leq Q_e\right] \geq 1 - \alpha, \quad \forall e \in E \tag{4.8}$$

$$P\left[\sum_{r_k \in R_e} b_{r_k}^{\uparrow} \leq B_e^{\uparrow}\right] \geq 1 - \alpha, \quad \forall e \in E \tag{4.9}$$

$$P\left[\sum_{r_k \in R_e} b_{r_k}^{\downarrow} \leq B_e^{\downarrow}\right] \geq 1 - \alpha, \quad \forall e \in E \tag{4.10}$$

A standard approach for solving such optimization problems is to transform the probabilistic constraints into equivalent deterministic ones. By doing so, the original linear stochastic constraint may no longer remain linear after the transformation. We use a similar approach as in [41] to transform the probabilistic constraints to their deterministic equivalents, thereby making them solvable by integer programming.

4.3.3 Transformation for known distributions

For the sake of simplicity and ease of illustration and to derive exact bounds, we first describe the case when the resource utilization of service requests follow a known probabilistic distribution. We present our steps assuming normal distributions for each random variable for each resource dimension. Let us assume each of $s_{r_i}, c_{r_i}, b^{\uparrow}_{r_i}$ and $b^{\downarrow}_{r_i}$ of any service $r_i \in R$ follows a normal distribution with corresponding means $\mu^s_{r_i}, \mu^c_{r_i}, \mu^{b^{\uparrow}}_{r_i}, \mu^{b^{\downarrow}}_{r_i}$ and standard deviation $\sigma^s_{r_i}, \sigma^c_{r_i}, \sigma^{b^{\uparrow}}_{r_i}, \sigma^{b^{\downarrow}}_{r_i}$ respectively. We illustrate the transformation using the CPU dimension. We rewrite Equation 4.7 as:

$$P\left[X^c_e \leq C_e\right] \geq 1-\alpha,\ \forall e \in E \tag{4.11}$$

where $X^c_e = \sum_{r_k \in R_e} c_{r_k}$ is a random variable as well [9], denoting the aggregate CPU demand at an edge server, following a normal distribution with mean as $\eta^c_e = \sum_{r_k \in R_e} \mu^c_{r_k}$ and variance as $(\gamma^c_e)^2 = \sum_{r_k \in R_e} (\sigma^c_{r_k})^2$. Now,

$$P[X^c_e \leq C_e] \geq 1-\alpha \implies P\left[\tfrac{X^c_e-\eta^c_e}{\gamma^c_e} \leq \tfrac{C_e-\eta^c_e}{\gamma^c_e}\right] \geq 1-\alpha$$

Let $Z^c_e = \frac{C_e-\eta^c_e}{\gamma^c_e} = \phi(\frac{C_e-\eta^c_e}{\gamma^c_e})$, where $\phi(z) = P[Z \leq z] = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} \exp^{-z^2/2}\, dz$ is the cumulative distribution function of the variable Z. We therefore, have the following:

$$P\left[\tfrac{X^c_e-\eta^c_e}{\gamma^c_j} \leq \tfrac{C_e-\eta^c_e}{\gamma^c_e}\right] \geq 1-\alpha \implies \phi(\tfrac{C_e-\eta^c_e}{\gamma^c_e}) \geq 1-\alpha \implies \eta^c_e + \gamma^c_e \phi^{-1}(1-\alpha) \leq C_e,$$

where $\phi^{-1}(1-\alpha)$ is the $(1-\alpha)$-th quantile of the standard normal distribution. Thus, the transformed deterministic constraint formulation is as below:

$$\eta^c_e + \gamma^c_e \phi^{-1}(1-\alpha) \leq C_e,\ \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.12}$$

Similarly, the transformations for Equations 4.8, 4.9 and 4.10 are:

$$\eta^r_e + \gamma^r_e \phi^{-1}(1-\alpha) \leq Q_e,\ \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.13}$$

$$\eta^{b^{\uparrow}}_e + \gamma^{b^{\uparrow}}_e \phi^{-1}(1-\alpha) \leq B^{\uparrow}_e,\ \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.14}$$

$$\eta^{b^{\downarrow}}_e + \gamma^{b^{\downarrow}}_e \phi^{-1}(1-\alpha) \leq B^{\downarrow}_e,\ \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.15}$$

Finally, using the determinized constraints obtained above, the ILP formulation to maximize the number of users allocated on the edge can be formulated as below. We define the following:

$$x_{es} = \begin{cases} 1, & \text{If the service } s \text{ is placed at edge server } e \\ 0, & \text{Otherwise} \end{cases}$$

and $y_{ei} = \begin{cases} 1, & \text{If service request } r_i \in R \text{ is allocated to edge } e \\ 0, & \text{Otherwise} \end{cases}$

Objective:

$$Maximise \sum_{r_i \in R, e \in E} y_{ei} \tag{4.16}$$

Subject To:

- Integer Constraints:

$$x_{es} \in \{0,\ 1\} \quad : \ e \in E,\ s \in S \tag{4.17}$$

$$y_{ei} \in \{0,\ 1\} \quad : \ e \in E \cup \{E_c\},\ r_i \in R \tag{4.18}$$

- Coverage Constraint:

$$y_{ei} = 0, \quad \forall r_i \ \in \ R,\ e \notin \ E_{u(r_i)} \tag{4.19}$$

  $u(r_i)$ is the owner of request r_i and $E_{u(r_i)}$ is the set of edge servers covering user $u(r_i)$.

- Service Placement Constraint:

$$y_{ei} \leq x_{er_i}, \quad \forall e \in E,\ r_i \in R \tag{4.20}$$

  Note that each $r_i \in R$ refers to a service in S.

- Memory Constraint:

$$\sum_{r_i \in R_e} x_{er_i}\mu^s_{r_i} + \phi^{-1}(1-\alpha)\sqrt{\sum_{r_i \in R_e} x_{es}(\sigma^s_{r_i})^2} \leq Q_e, \quad \forall e \in E \tag{4.21}$$

- Computation Load Constraint:

$$\sum_{r_i \in R_e} y_{ei}\mu^c_{r_i} + \phi^{-1}(1-\alpha)\sqrt{\sum_{r_i \in R_e} y_{ei}(\sigma^c_{r_i})^2} \leq C_e, \quad \forall e \in E \tag{4.22}$$

- Bandwidth Constraint:

$$\sum_{r_i \in R_e} y_{ei}\mu^{b^\uparrow}_{r_i} + \phi^{-1}(1-\alpha)\sqrt{\sum_{r_i \in R_e} y_{ei}(\sigma^{b^\uparrow}_{r_i})^2} \leq B^\uparrow_e, \quad \forall e \in E \tag{4.23}$$

$$\sum_{r_i \in R_e} y_{ei}\mu^{b^\downarrow}_{r_i} + \phi^{-1}(1-\alpha)\sqrt{\sum_{r_i \in R_e} y_{ei}(\sigma^{b^\downarrow}_{r_i})^2} \leq B^\downarrow_e, \quad \forall e \in E \tag{4.24}$$

- User-Server Mapping:

$$\sum_{e \in E_u \cup E_c} y_{ei} = 1, \quad \forall r_i \in R \tag{4.25}$$

The integer program formulation aims to maximize the number of users allocated to the edge, as in equation 4.16 along with other constraints. Users should only be allocated to an edge server when within the coverage of that edge server, this condition is expressed in constraint 4.19. A service provisioned has to be hosted on the edge server, as expressed in constraint 4.20. A single user should not be allocated to more than one edge server, as in Equation 4.25. The constraints in Equations 4.21, 4.22, 4.23 and 4.24 are deterministic ones which bound the overflow probability on each edge server. A solution satisfying all constraints for a given α is an allocation for a user request to an edge server, such that the overflow probability on the aggregate memory demand of each request allocated to an edge server is bounded for each dimension.

4.3.4 Handling Unknown Distributions

We now discuss the case where the probabilistic distribution of the resource parameters is unknown. In this case, we use the Chebyshev's inequality [9] for analysis. If X is a random variable with mean μ and variance $\sigma^2(\neq 0)$ and t is a positive real number, one-sided Chebyshev's inequality (Cantelli's inequality) can be stated as: $P[\frac{X-\mu}{\sigma} > t] \leq \frac{1}{1+t^2}$ as mentioned in Chapter 2, Section 2.1.4. Let us assume the resource requirements $s_{r_i}, c_{r_i}, b^{\uparrow}_{r_i}$ and $b^{\downarrow}_{r_i}$ of any service request $r_i \in R$ requested by any user $u \in U$ follows an unknown distribution with mean $\mu^s_{r_i}, \mu^c_{r_i}, \mu^{b^{\uparrow}}_{r_i}, \mu^{b^{\downarrow}}_{r_i}$ and standard deviation $\sigma^s_{r_i}, \sigma^c_{r_i}, \sigma^{b^{\uparrow}}_{r_i}, \sigma^{b^{\downarrow}}_{r_i}$ respectively. In this case as earlier, we use $X^c_e = \sum_{r_k \in R_e} c_{r_k}$ to denote a random variable, denoting the aggregate memory demand at an edge server, following an unknown distribution with mean as $\eta^c_e = \sum_{r_k \in R_e} \mu^c_{r_k}$ and variance as $(\gamma^c_e)^2 = \sum_{r_k \in R_e} (\sigma^c_{r_k})^2$.

Now, $P[X^c_e \leq C_e] \geq 1 - \alpha$

$$\implies P\left[\frac{X^c_e - \eta^c_e}{\gamma^c_e} \leq \frac{C_e - \eta^c_e}{\gamma^c_e}\right] \geq 1 - \alpha \implies P\left[\frac{X^c_e - \eta^c_e}{\gamma^c_e} > \frac{C_e - \eta^c_e}{\gamma^c_e}\right] \leq \alpha.$$

Using one-sided Chebyshev's inequality with $t = \frac{C_e - \eta^c_e}{\gamma^c_e}$

$$\implies P\left[\frac{X^c_e - \eta^c_e}{\gamma^c_e} > \frac{C_e - \eta^c_e}{\gamma^c_e}\right] \leq \frac{1}{1+(\frac{C_e - \eta^c_e}{\gamma^c_e})^2} \leq \alpha \implies \eta^c_e + \gamma^c_e \sqrt{\frac{1-\alpha}{\alpha}} \leq C_e,$$

Thus, the transformed deterministic constraint formulation of Equation 4.7 is as below:

$$\eta^c_e + \gamma^c_e \sqrt{\frac{1-\alpha}{\alpha}} \leq C_e, \quad \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.26}$$

Similarly, the transformations for Equations 4.8, 4.9 and 4.10 are:

$$\eta^r_e + \gamma^r_e \sqrt{\frac{1-\alpha}{\alpha}} \leq Q_e, \quad \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.27}$$

$$\eta^{b^{\uparrow}}_e + \gamma^{b^{\uparrow}}_e \sqrt{\frac{1-\alpha}{\alpha}} \leq B^{\uparrow}_e, \quad \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.28}$$

$$\eta^{b^{\downarrow}}_e + \gamma^{b^{\downarrow}}_e \sqrt{\frac{1-\alpha}{\alpha}} \leq B^{\downarrow}_e, \quad \forall e \in E,\ 0 \leq \alpha \leq 1 \tag{4.29}$$

The above derived Equations 4.26, 4.27, 4.28 and 4.29 replace the Equations 4.12, 4.13, 4.14 and 4.15 respectively while formulating deterministic integer programming constraints with unknown distributions. We now present our experiments in the section below.

4.4 Experiments and Analysis of Results

4.4.1 Experimental Setup

We conducted two sets of experiments, one with the EUA real world data set [31] and another with the simulated MEC environment representative of large-scale MEC scenarios. For the simulated MEC

environment we consider a setup similar to [49]. We consider a region of area $500m \times 500m$. We vary the number of servers as 8 and 10 by fixing the number of services to 5. For each server, we vary the number of users from 800 to 1600 at an interval of 200 users. The edge servers and users are placed randomly inside the MEC environment region with each edge server having the coverage range of 150m. The overflow probability α is set to 0.15 for the experiment. The time out for all the algorithms is set to $250s$. The reported results are obtained by averaging over 50 repeated experiments. The resources of each server and requirements of service requests are represented as [C, R, BU, BD], for example, the resource availability of each edge server is set to [20vCPUs, 250GB, 150Mbps, 300Mbps].

Services	Resources							
	Computation		Memory		Uplink Bandwidth		Downlink Bandwidth	
	mean	std	mean	std	mean	std	mean	std
Video Streaming	0	0	1-4	0-1.3	0	0	1-8.5	0-2.8
Face Recognition	0.4-1.6	0-0.5	1-5	0-1.6	1-4	0-1.3	0	0
File Compression	0.05-0.2	0-0.06	0.02	0	1-4	0-1.3	0.25-1	0
AR	0.3-1.2	0-0.4	1-10	0-3.3	1-4	0-1.3	0.25-1	0-0.3
ML Inference	0.35-1.4	0-0.6	1-10	0-3.3	0.5-2	0-0.6	0.1	0
Online Game	1-3	0-0.1	1-3	0-0.1	1-2	0-0.66	1-2	0-0.66

Table 4.2: Parameters for Services used in the experiment

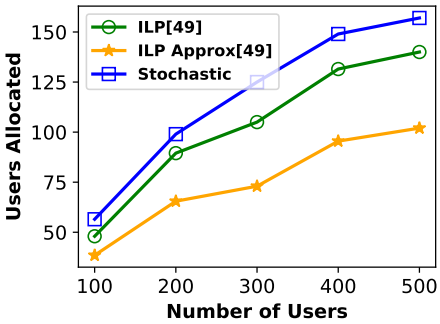
The experimental setup for the EUA data set [31] follows an approach similar to the simulated MEC setup, however, the edge server and user locations are from the real world data set. The experiment using the EUA dataset is done using the values considering average and maximum resource requirements. We carry out several sets of experiments varying the number of servers, services and resource availability of the servers. The services requested by users are taken from the zipf distribution [49] initially assuming the number of services equal to the number of users as in [49]. Each service request is mapped randomly to one of the services shown in Table 4.2. Each set of services in Table 4.2 presents some predetermined range of values from which the actual service values are assigned randomly. For example, if a service s is mapped to the Augmented Reality service with mean as 1-4 for memory, the memory requirement value μ_s^r for that service will be assigned randomly from the range 1 to 4.

The services used in this experiment, as shown in Table 4.2 are representative of widely used services hosted on the edge servers [49]. The Video Streaming service is characterized by high download bandwidth requirements. Note that rendering the video is carried out on the user device, hence, there is no computation requirement on the edge server associated with such a service. The Face Recognition service uses uplink bandwidth, memory and computation for uploading the image to the edge server, processing the image while additionally utilizing memory for the Machine Learning Inference models for face recognition at an Edge Server. The File Compression service, on the other hand, utilizes uplink bandwidth to upload the data, consumes runtime memory for processing the data and high computational resource to compress the file. The Augmented Reality application is resource intensive and utilizes all such resources. The ML Inference service used by IoT sensors uses memory to store the results of the inference algorithm while also consuming uplink bandwidth to upload sensor data to edge servers. The Online Game service, similar to Augmented Reality, is resource intensive and utilizes all resources to provide a smooth immersive experience to users.

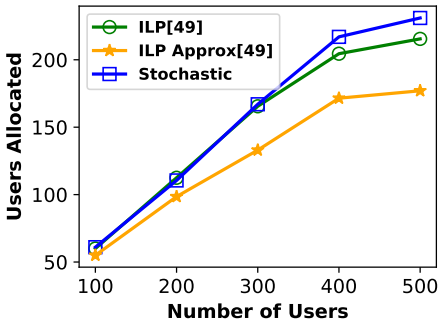
We compare the effectiveness of our stochastic approach with the ILP and approximation algorithm in [49]. We consider two cases for each set of experiments, (i) where the maximum resource utilization of different services are used in conjunction with the ILP and approximation approaches and (ii)

where the average resource utilization of different services are utilized for the ILP and approximation algorithms. The maximum value used for the deterministic ILP in [49] is computed using the maximum value of the 1000 random samples drawn from the normal distribution using mean and variances for different services accordingly as given in Table 4.2. Similarly, the average value is estimated by averaging over 1000 random samples from the normal distribution using respective mean and variance of the services. To demonstrate the scenarios when the distribution is known, the resource utilization of the services are assumed to follow the normal distribution using mean and variance of the respective services as outlined in Table 4.2. For unknown distributions, the mean and variance as outlined in Table 4.2 are re-utilized, however, without a normal distribution being assumed. The overflow / under-use are calculated by averaging over 100 repeated evaluations. All experiments are conducted on an Intel Core i5-8250U processor with 8GB RAM. We use the IBM ILOG CPLEX suite for optimization [1].

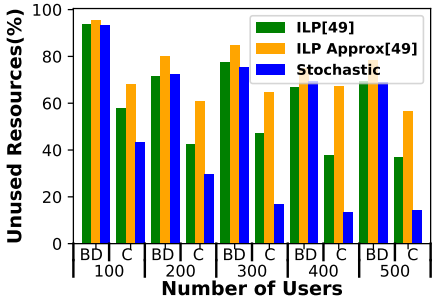
To demonstrate the effectiveness of our approach, the number of users allocated and percentage of resource overflow / under-use are compared. The overflow / under-use is calculated by computing the difference in total resources utilized by the services from the resources available on the edge server due to the user-server-service allocation. In the following sub-sections, we discuss several different experiment sessions such as varying resources, the number of edge servers and number of services on the edge servers when utilizing the maximum value of resource requirements and the average value of resource requirements.



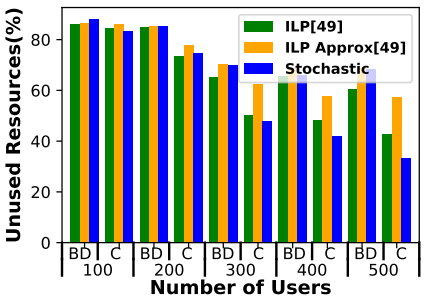
(a) Allocation for [5, 15, 25, 35]



(b) Allocation for [15, 30, 30, 40]



(c) Unused% for [5, 15, 25, 35]

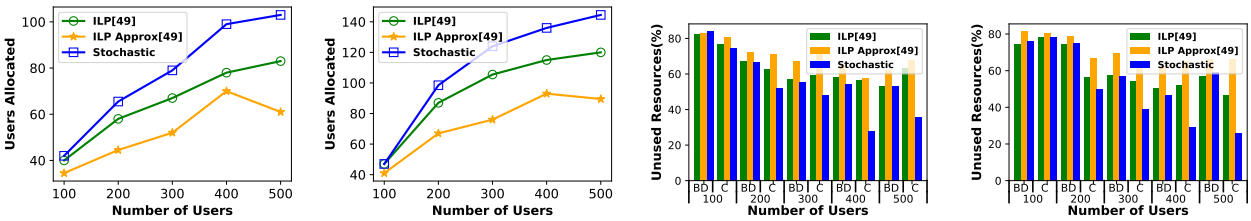


(d) Unused% for [15, 30, 30, 40]

Figure 4.2: Varying Resources with E = 20, S=6 for Maximum Case on EUA Data-set under Normal Distribution

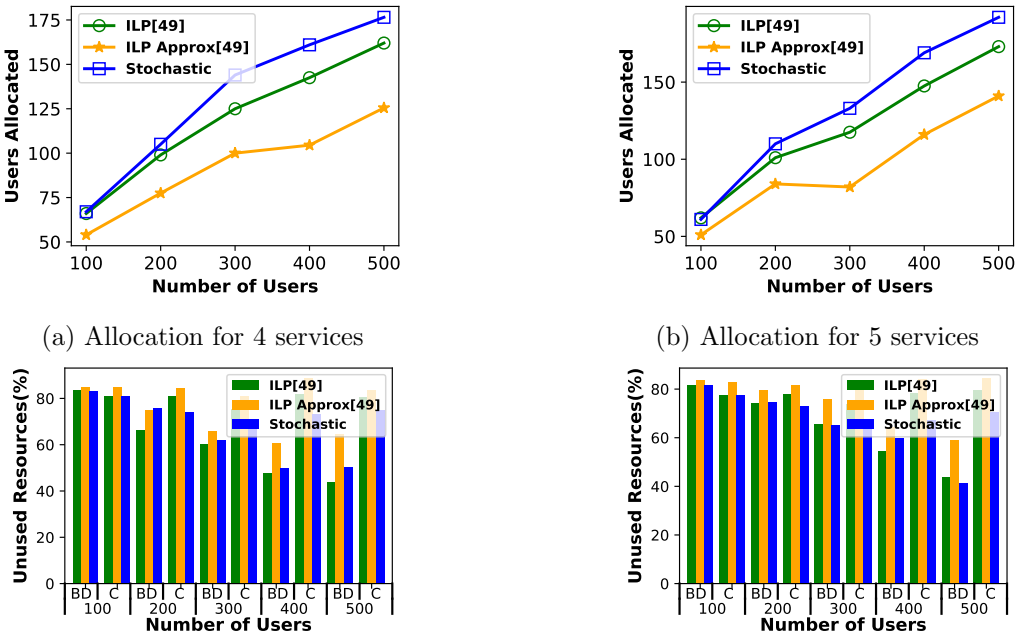
4.4.2 Varying Server Resources

Figures 4.2a and 4.2b depict the number of users allocated when varying the resource availability of the edge servers. The number of servers is fixed at 20 while the number of services is fixed at 6. In both scenarios, there is an increase in the number of users allocated on the severs by our



(a) Allocation for E = 10 (b) Allocation for E = 20 (c) Unused% for E = 10 (d) Unused% for E = 20

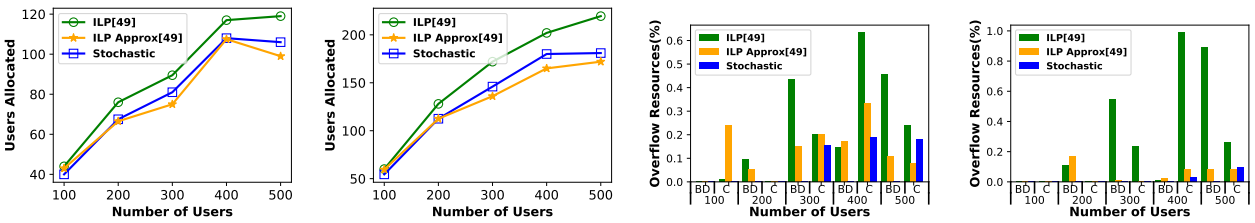
Figure 4.3: Varying Servers with [10, 20, 15, 30], S = 6 for Maximum case on EUA Data-Set under Normal Distribution



(a) Allocation for 4 services (b) Allocation for 5 services

(c) Unused% for 4 services (d) Unused% for 5 services

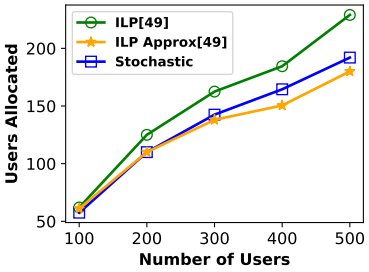
Figure 4.4: Varying Services with [10, 20, 15, 30], E = 20 for Maximum Case on EUA Data-Set under Normal Distribution



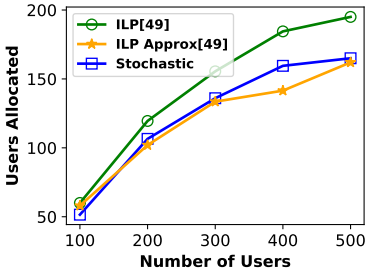
(a) Allocation for E = 10 (b) Allocation for E = 20 (c) Overflow% for E = 10 (d) Overflow% for E = 20

Figure 4.5: Varying Servers with [10, 20, 15, 30], S = 6 for Average Case on EUA Data-Set under Normal Distribution

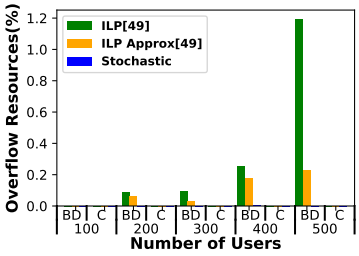
stochastic ILP over the ILP and the approximation approaches as in [49]. Thus, a fewer number of users are sent to the cloud. The greatest variation is observed when number of users is 500 where the stochastic ILP provides substantial improvement in number of users allocated over the other approaches. The difference however is not so prominent in the other scenarios for lesser number of users. Figures 4.2c and 4.2d depict the percentage of unused resources in such scenarios. CPU and



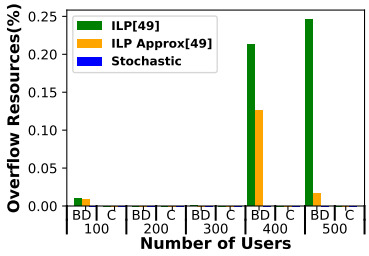
(a) Allocation for 4 services



(b) Allocation for 5 services

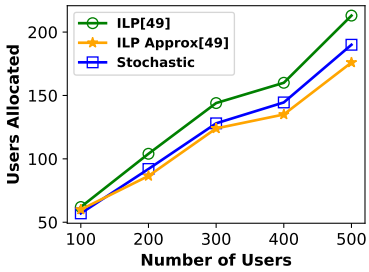


(c) Overflow% for 4 services

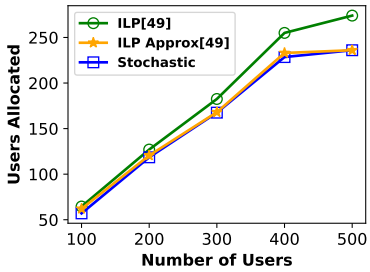


(d) Overflow% for 5 services

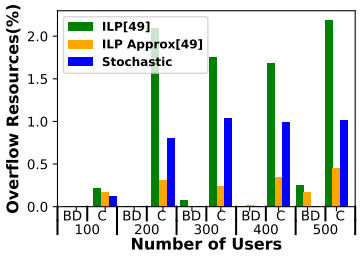
Figure 4.6: Varying Services with [10, 20, 15, 30], E = 20 for Average Case on EUA Data-Set under Normal Distribution



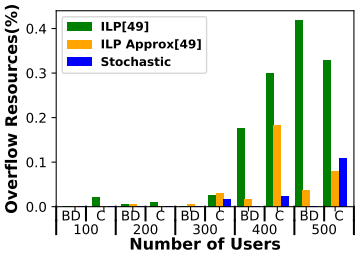
(a) Allocation for [5, 15, 25, 35]



(b) Allocation for [15, 30, 30, 40]



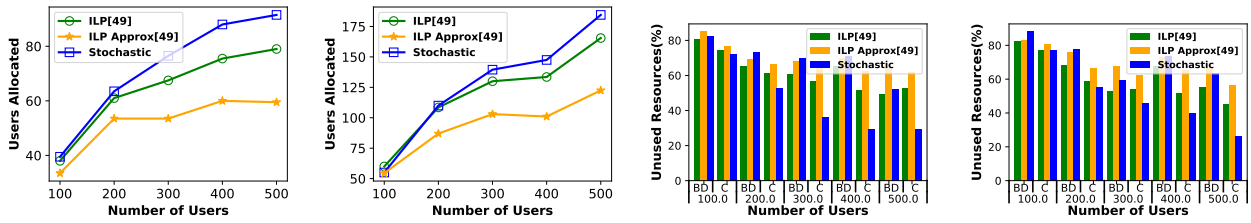
(c) Overflow% for [5, 15, 25, 35]



(d) Overflow% for [15, 30, 30, 40]

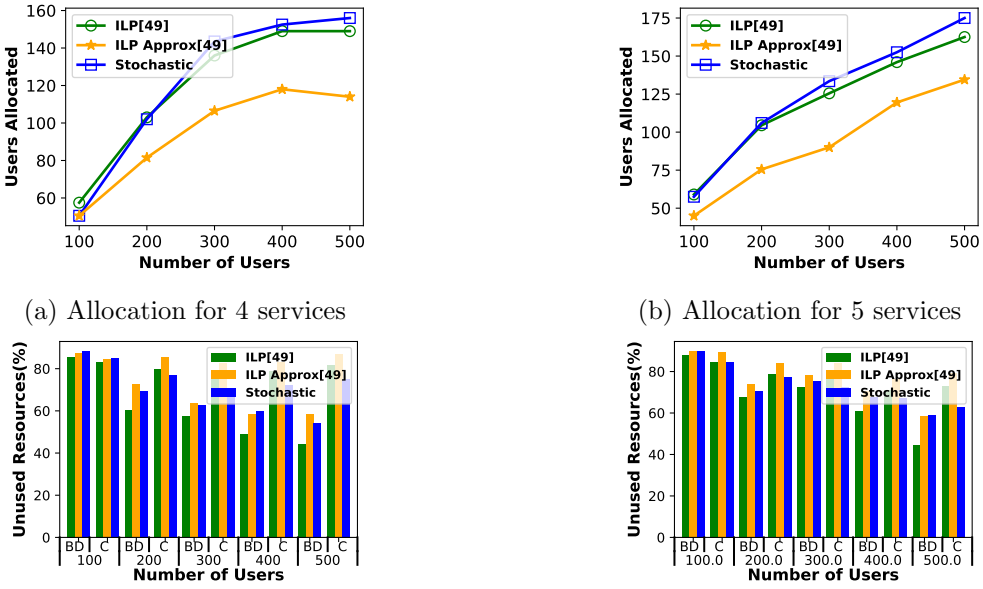
Figure 4.7: Varying Resources with E = 20, S=6 for Average Case on EUA Data-Set under Normal Distribution

Downlink Bandwidth (BD) are depicted while Storage and Uplink Bandwidth omitted for brevity. It is interesting to note that in all these scenarios, there is no definite increasing / decreasing pattern with increase in the number of users. Such a scenario is a consequence of the fact that resource utilization is a characteristic of the nature of the service requests being allocated to an MEC server. Hence, depending on the type of service requests, the number of allocated users vary. Storage and Uplink Bandwidth follow similar characteristics. Such observations depict the fact that resource availability plays a key role in determining the effectiveness of the approaches.



(a) Allocation for E = 10 (b) Allocation for E = 20 (c) Unused% for E = 10 (d) Unused% for E = 20

Figure 4.8: Varying Servers with [10, 20, 15, 30], S = 6 for Maximum case on EUA Data-Set under Unknown Distribution



(a) Allocation for 4 services

(b) Allocation for 5 services

(c) Unused% for 4 services

(d) Unused% for 5 services

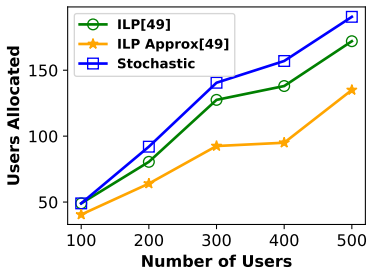
Figure 4.9: Varying Services with [10, 20, 15, 30], E = 20 for Maximum Case on EUA Data under Unknown Distribution

4.4.3 Varying Number of Edge Servers

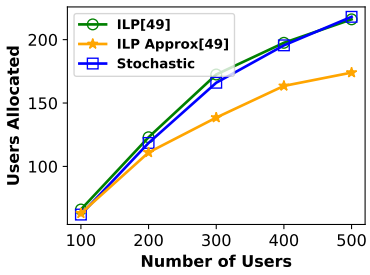
Figures 4.3a and 4.3b depict the number of users allocated when varying the number of edge servers. The resource availability of servers is kept fixed at [10GHz, 20GBs, 15Mbps, 30Mbps] while the number of services is fixed at 6. The stochastic ILP once again, obtains higher gains when the number of users is 500. Such scenarios is in concordance with the earlier scenario of resource variability since varying the number of edge servers while keeping the resource availability of each server fixed is akin to varying resource availability. Thus, as depicted by the two experiments, such scenarios conform to each other. Additionally, as the number of servers is increased, the greater is the availability of server resources. Hence in such scenarios, the stochastic ILP, the ILP and approximation approaches perform equally well. The stochastic ILP depicts considerable gains in resource constrained scenarios.

4.4.4 Varying Number of Services

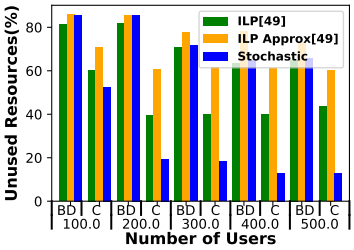
Figures 4.4a and 4.4b depict the number of users allocated when varying the number of services available on the edge servers. As in the earlier set of experiments, the resource availability of servers is kept fixed at [10GHz, 20GBs, 15Mbps, 30Mbps], while the number of servers is fixed at 20. With



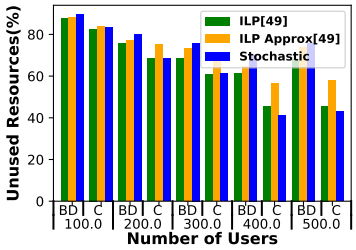
(a) Allocation for [5, 15, 25, 35]



(b) Allocation for [15, 30, 30, 40]

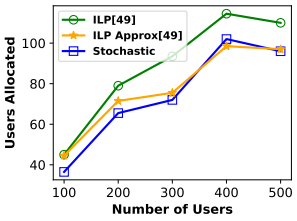


(c) Unused% for [5, 15, 25, 35]

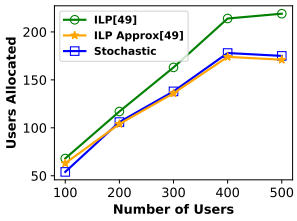


(d) Unused% for [15, 30, 30, 40]

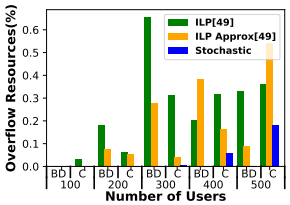
Figure 4.10: Varying Resources with E = 20, S=6 for Maximum Case on EUA Data-Set under Unknown Distribution



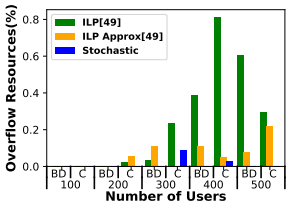
(a) Allocation for E = 10



(b) Allocation for E = 20



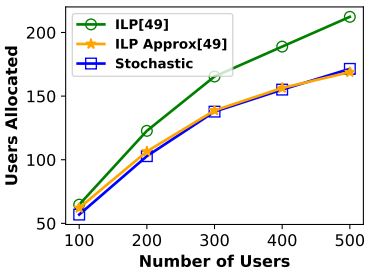
(c) Overflow% for E = 10



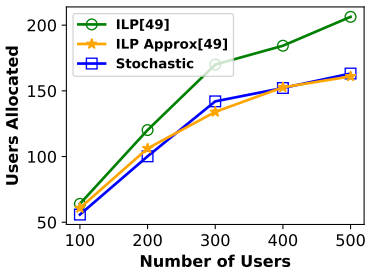
(d) Overflow% for E = 20

Figure 4.11: Varying Servers with [10, 20, 15, 30], S = 6 for Average Case on EUA Data-Set under Unknown Distribution

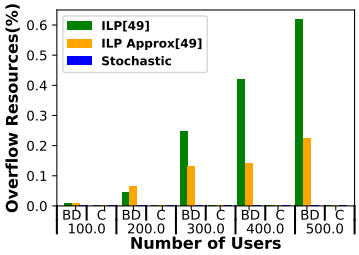
an increasing number of services, coupled with increasing number of users, the stochastic ILP gains substantially over traditional approaches. Our stochastic formulation is thus seen to cater well to scenarios where the number of users is large. Figures 4.5, 4.6 and 4.7 depict the results on the same set of experiments as in Figures 4.2, 4.3 and 4.4, however, with the average values of service resource requirements being utilized for the ILP and approximation approaches. In such scenarios, the difference between the number of users allocated by the stochastic approach and traditional ILP and approximation dwindles down. However, in such scenarios, as can be inferred from Figures 4.5c and 4.5d, the number of overflows are high. Such a scenario occurs since considering an average value for service resource utilization leads to an under-allocation of resources in several scenarios when at run-time, the actual resources consumed is higher than what was estimated. Our stochastic formulation, on the other hand, leads to a much lower number of such encountered scenarios since our approach directly encapsulates such variational behaviour. On the other hand, utilizing the maximum value of service resource utilization leads to an under utilization of available server resources while avoiding overflow scenarios thereby depicting the trade-offs involved. A similar trend is observed with Figures 4.6c, 4.6d, 4.7c and 4.7d demonstrating the robustness of our stochastic characterization towards workload variation. Figures 4.6c and 4.6d depict the effectiveness of our approach when the number of users are on the higher side. With 400 and 500 users, the unused resource percentage is negligible as compared to the other approaches.



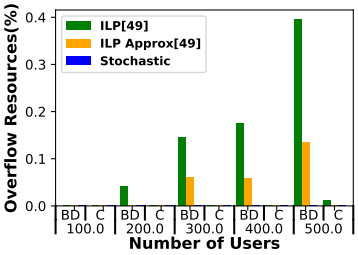
(a) Allocation for 4 services



(b) Allocation for 5 services

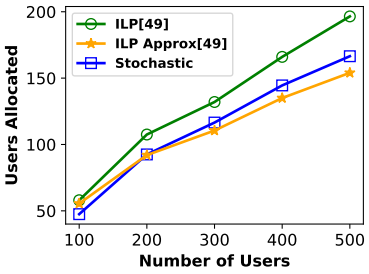


(c) Overflow% for 4 services

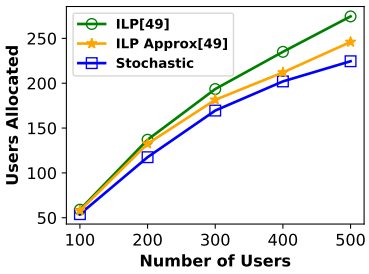


(d) Overflow% for 5 services

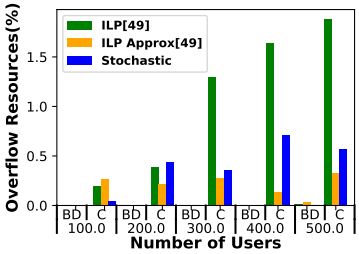
Figure 4.12: Varying Services with [10, 20, 15, 30], E = 20 for Average Case on EUA Data-Set under Unknown Distribution



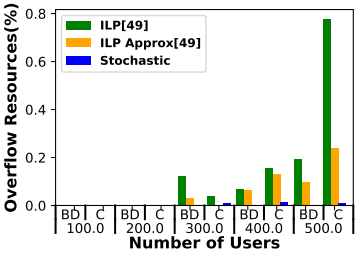
(a) Allocation for [5, 15, 25, 35]



(b) Allocation for [15, 30, 30, 40]



(c) Overflow% for [5,15,25,35]



(d) Overflow% for [15,30,30,40]

Figure 4.13: Varying Resources with E = 20, S=6 for Average Case on EUA Data-Set under Unknown Distribution

4.4.5 Unknown Distributions

We additionally repeat the same experimental scenarios described earlier, however, with the relaxation of the assumption that the resource requirements follow a normal distribution. Figures 4.8 - 4.10 depict the results obtained with the stochastic characterization of an unknown service request distribution when the maximum resource requirements are used for the ILP and approximation approaches while Figures 4.11 - 4.13 depicts the plots with average resource utilization. A similar set of behaviour is

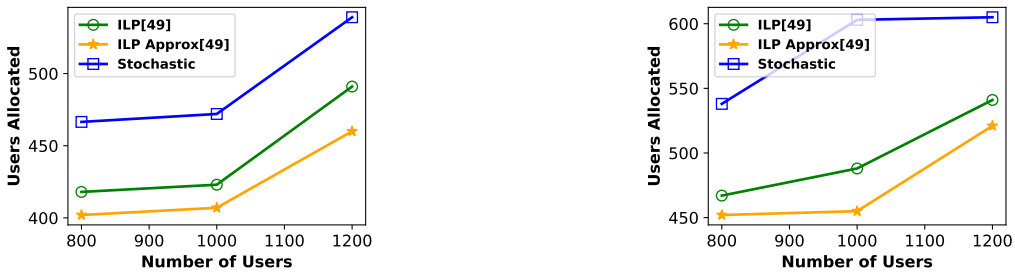
(a) 4 services (b) 5 services

Figure 4.14: Extra latencies incurred due to overflow with [10, 20, 15, 30] and E=20 for Average Case on EUA Data-set under Unknown Distribution

observed where our stochastic formulation performs better as opposed to service workload agnostic approaches. For unknown distributions, when the ILP and approximation approaches utilize the maximum resource requirements of services, the number of users allocated by our stochastic approach is higher as compared to service workload agnostic approaches. In scenarios when the average resource utilization is considered, a similar trend is observed as in case of the normal distribution with a reduction in the difference between the number of users allocated. However, overflow violations are higher on average in such scenarios.

4.4.6 Large Scale Scenarios

Figure 4.15 depicts scenarios generated on random user and server locations as used in [49]. The large scale scenarios follow identical patterns as on the EUA dataset with no definite increase/decrease pattern. However, with a large number of users, our stochastic approach prominently allocates more number of users as compared to the ILP and approximation approaches. The percentage of unused resources is not depicted for such scenarios for brevity, however, they follow a similar pattern as observed with the EUA dataset.



(a) E = 8 (b) E = 10

Figure 4.15: Allocation for large scale environment with 6 services

4.4.7 Handling Overflow Scenarios

We handle overflow scenarios by re-allocating users involved in the overflow to the cloud server, with additional access latencies. The additional latencies to the cloud is taken as $112ms$, as shown in [76] measured as the real world round-trip latency to a public cloud provider. Figure 4.14 illustrates the extra latencies incurred due to overflow for the set-up as in Figure 4.12. The access latencies vary in other scenarios, however, on average, the stochastic approach outperforms the traditional approaches.

The stochastic optimization approach performs well in comparison to traditional approaches which do not take workload variations into account but performs especially well in large scale scenarios where a greater variation in the number of allocated users at edge servers is observed. The availability of resources on edge servers play a crucial part in such scenarios. Even in case of overflow, the latencies incurred as a result of re-allocating requests to the back-end cloud is lower for our stochastic approach as compared to traditional approaches.

4.5 Conclusion

In this chapter, we propose a joint service allocation and placement policy taking into account the stochastic nature of service workloads. We perform extensive experiments on real-world datasets to demonstrate the effectiveness of our approach. However, the proposed model relies on the mathematical formulation of the MEC system which is often challenging to formulate accurately. In the next chapter, we propose a data-driven learning-based solution that implicitly handles the workload variation and non-linearity in the system.

Chapter 5

User Service Server Allocation using Deep Reinforcement Learning

5.1 Introduction

The workload fluctuation aware policy, discussed in the previous chapter uses stochastic parameters, mean and standard deviation, to model service resource utilization. For the method to work, the parameters themselves need to be obtained from service execution footprints which is often highly dynamic since the execution data obtained at one instant of time may significantly vary at another instant. Furthermore, we need to specify the stochastic distribution in the model to get the best results. In reality, modelling a highly non-linear system like service resources utilization is quite challenging. The model assumed linear dependence with some stochasticity on resource utilization of services hosted on the edge server. In addition, many other traditional prior works assume linear dependence of resource utilization on the number of service requests provisioned on the edge server. Nowadays, a large number of applications are being integrated with ML-based solutions which often need GPUs for fast execution. Modelling the service resources utilization on GPU is far more mathematically challenging. We show via experiments on both CPUs and GPUs that the resource-service-usage relationship is usually highly non-linear, i.e. the resource utilization by services does not scale linearly if the number of requests grows as shown in [27,37,51,53,61] using the Google cluster trace dataset [54]. The nonlinearity arises due to the effect of various internal system attributes such as software/hardware architecture, operating system policies, number of cores, varying nature of service workloads in CPU/GPU, service invocations pattern etc.

In deterministic approaches, the total resource utilization at each server is dependent on the cumulative sum of the resource utilization footprints of the service requests [30,31,47]. In a real deployment, the amount of resources that may be utilized by any service during execution is highly dynamic, which is often difficult to model mathematically. Generally, the resource utilization of services in prior research works are taken as the mean or maximum based on the records of service execution. However,

This work is accepted as:

- Panda S.P., Banerjee A. and Bhattacharya A., User Allocation in Mobile Edge Computing: A Deep Reinforcement Learning Approach. Accepted for Presentation at the 28th IEEE International Conference on Web Services (ICWS) 2021.

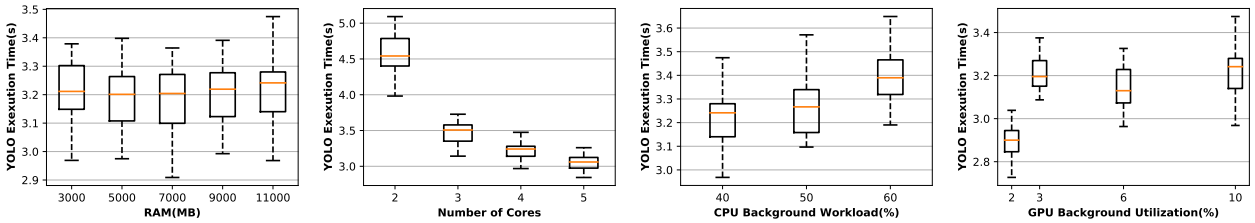


Figure 5.1: Non-Linear relationship between different resource attributes and YOLO execution time (a) varying RAM (b) varying number of cores (c) varying CPU workload (d) varying GPU utilization

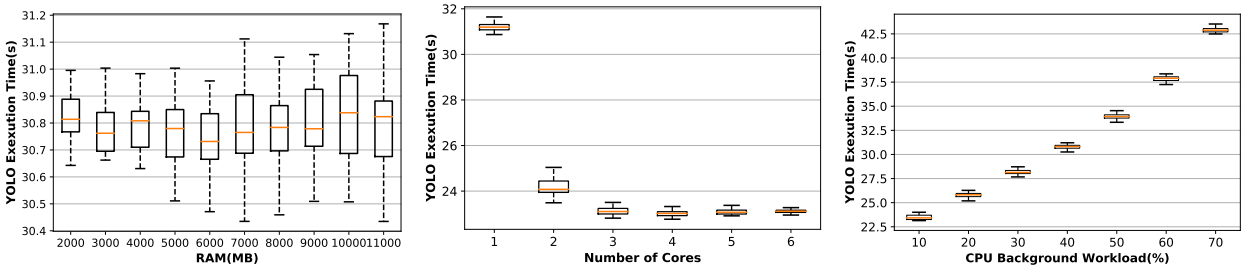


Figure 5.2: Non-Linear relationship between different resource attributes and YOLO execution time using only CPU (a) varying RAM (b) varying number of cores (c) varying CPU workload

these approaches lead to under-use or overflow of the edge servers due to inappropriate allocation. In this work, we aim to learn the resource utilization of services using machine learning instead of just assuming average or maximum values from past execution records. While utilizing supervised models is one possible way, they require an extensive amount of data for training, and cannot adapt if the recorded data at some instance of time changes in future. Thus, we utilize a DRL [39] based approach. DRL learns the resource allocation based on experiences incrementally instead of using a training dataset obtained a priori [38], which makes the DRL based framework trainable on the device itself.

In this chapter, we propose a DRL framework in which the DRL agent learns the system environment of edge servers i.e. it learns the number of users that can be served at a particular edge server under the constraint of a defined service latency threshold. Rather than formulating a complex mathematical model of the system, the DRL agents inherently learn non-linear dependencies directly from the edge server by observing system parameters over a while. The DRL agents do not need a dataset to perform the training task, hence can be trained online directly on the edge. We propose to use the DRL for on-device training to learn the edge system dynamics, thereby, reducing the overhead of the requirement of the training dataset. We propose an algorithm to obtain the user allocation policy from the trained DRL agent.

We have evaluated our approach using a real world dataset [31], which consists of locations of users and edge servers at CBD, City of Melbourne, Australia. We have compared the number of users allocated with the proposed method with two deterministic baseline approaches: (a) the ILP approach inspired from works in [30] [31] and (b) a Greedy solution to the ILP approach based on user allocation to servers in the respective nearest neighborhood. Our approach outperforms the traditional linear approach by approximately 10% with 500 users and 50 edge servers in an MEC environment.

The rest of this chapter is organized as follows. We illustrate a motivating scenario for this work in Section 5.2. In Section 5.3, we provide a short description of DRL and formulate our DRL agent to solve the EUA problem. We then provide two deterministic baseline approaches in Section 5.4. Subsequently, in Section 5.5 we discuss experiments and results obtained using our proposed approach.

Finally, we conclude the chapter in Section 5.6

5.2 Motivation

In this section, we first show via experiments how the assumption on the linear relationship between number of service requests hosted and the cumulative additive resource usage breaks in real service deployments. We then use a motivating example to demonstrate how these assumptions lead to incorrect allocations of users to edge devices.

5.2.1 Observations to Verify Assumptions

We first observe the service execution times of a widely used object detection application YOLO [52] in Figures 1 and 2 both using and without using a GPU respectively, to process an image. We run the experiments on a machine with Intel(R) Xeon(R) CPU E5-1650 v4 processor, 64GB RAM and Quadro P4000 8GB GPU (further details in Section 5.5). By default, we restrict the amount of RAM at 11000 MB, number of cores equal to 4, CPU background workload at 40% and GPU background workload at 10%. We then vary only a single parameter for each experiment, where the parameters are (a) available RAM, (b) number of available cores, (c) CPU background workload, and (d) GPU background workload. We repeat the experiments a total of 20 times and show the execution times in box plots.

We note that modeling the execution times of services is challenging due to the following factors:

- **Non-linear relationship between available processor resources and execution time:** The relationship between the execution time and the parameter is non-linear. For example, in Figure 5.1(b), we find that reducing the number of available cores increases the execution time by a much higher factor in contrast to the situation when the available number of cores is small. Similarly, increasing the background workload of CPU and GPU slows down execution much more if the workload is already high. Moreover, there are significant variations in the execution time, making them difficult to model directly. The optimization models in prior research do not accommodate these factors.

- **Variation Across Time:** We find in Figures 1 and 2 that there is a substantial difference in execution times *even with identical configurations on the same machines.* For example, in Figure 1(a), the execution times vary from 2.82s to 3.47s. This occurs because execution of services depend on multiple hidden parameters, such as service invocation patterns, CPU temperature, Operating Systems scheduling, disk characteristics, etc. Since most modeling techniques utilize deterministic values of execution times, it can lead to problems (overflow / underflow) in allocation of users to edge devices.

- **Variation Across Services:** Each service invocation has its own pattern of execution time. This makes the task of modeling the service execution times quite difficult. We show in Table 1 how the execution time of Yolo on e_1 increases approximately linearly with increase in the number of users, but that of MobileNet on e_1 is non-linear.

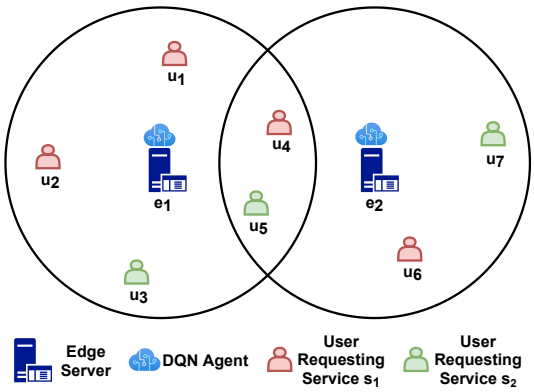


Figure 5.3: Representative MEC Server Allocation Scenario

5.2.2 A Motivating Example

Consider a simple scenario with seven mobile users $u_1, u_2 \dots u_7$ and two edge servers e_1 and e_2 as illustrated in Figure 5.3. Each user is requesting one of the two services s_1 and s_2 available on the edge servers. The users u_1, u_2, u_4 and u_6 are requesting the service s_1, whereas, remaining users are requesting for service s_2. For this example, the service s_1 corresponds to the YOLO [52] application and service s_2 corresponds to the MobileNetV2 [56] application. Each edge server is represented by a resource vector which represents the resources available and status of the edge server for service execution. We use the resource vector given as $\langle$*RAM (MB), Number of Cores, CPU Background Workload%, GPU Utilized%*$\rangle$ which consists of four system attributes that majorly impact service performance on an edge server. In this example, the resource vectors of edge server e_1 and e_2 are taken as $\langle 15000, 8, 60\%, 10\% \rangle$ and $\langle 6000, 4, 40\%, 6\% \rangle$ respectively. We now explain how using a deterministic approach leads to incorrect allocations.

Allocation with Deterministic Approaches: For allocation of users to edge servers with constraints on latency threshold, a deterministic value for service execution must be determined from the historical execution footprint. The deterministic service execution time for a service can be obtained from the execution footprint using approaches like averaging, or computation of median, maximum or regression. If we use a simple linear approach to find deterministic service execution time, for edge server e_1, the execution time for a single user request for service s_1 is 3.12s. Linearly interpolating this value for 4 users gives us an execution time of $12.48s$. However, in real-world execution, the execution time for 4 users is $3.46s$ as shown in Table 5.1. Let us assume that we are given a latency threshold of $6.5s$, i.e. the users should be allocated in such a way that their execution finishes in $6.5s$. A deterministic approach considering only the execution time of a single request for services will produce an allocation of u_1, u_2 and u_3 to e_1. Note that only two users can be served for service s_1 as each will take execution time of $3.12s$ producing a total of $6.24s$. Similarly, only one user can be served for service s_2 as it takes $6.32s$ of execution time for a single service. Thus, the total number of users we are able to allocate using the deterministic approach is equal to 3.

Potential of Data-driven Allocation Approach: As illustrated in Table 5.1, the execution time of four users running YOLO is below the latency threshold of $6.5s$. Thus, it was actually possible to allocate the users u_1, u_2, u_4 on e_1, as it only takes $3.35s$. Furthermore, it is also possible to accomodate u_5 and u_7 on e_2 as two users only take a total of $6.12s$. Thus, we are able to allocate a total of 5 users (i.e., 2 more users than the deterministic approach) using the data-driven approach, due to a more accurate modeling of resource utilization.

Users	YOLO on e1	YOLO on e2	MobileNet on e1	MobileNet on e2
1	3.12	3.26	6.32	6.03
2	3.23	3.29	6.40	6.12
3	3.35	3.37	6.42	6.18
4	3.46	3.50	6.54	6.26

Table 5.1: Service execution times of YOLO and MobileNetV2 on edge servers e_1 and e_2

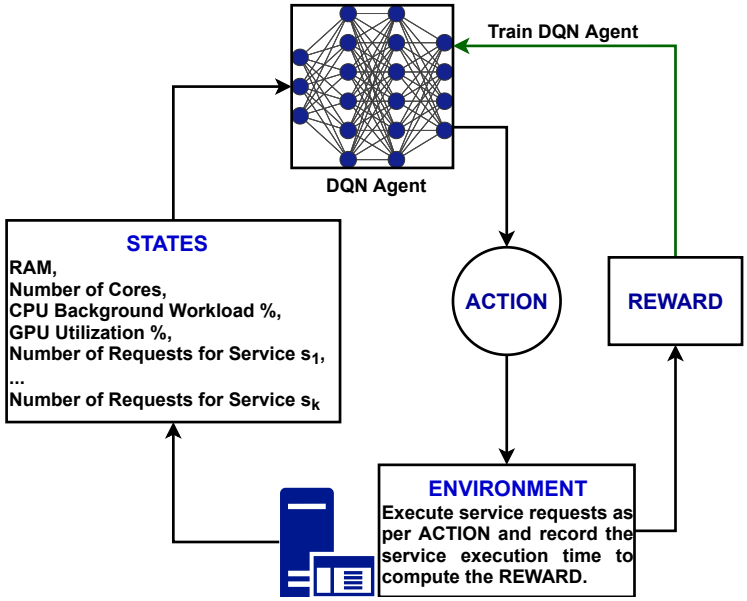


Figure 5.4: Illustration of Reinforcement Learning Framework

5.3 Allocation with Reinforcement Learning

The MEC environment comprises of edge servers denoted as $E = \{e_1, e_2, \ldots e_j\}$, where each edge server e_j has a coverage radius of r_j, the mobile users located within the coverage radius of an edge server can request for services hosted on that server. A set of users $U = \{u_1, u_2, \ldots u_i\}$ may request for services in set $S = \{s_1, s_2, \ldots s_k\}$ hosted on an edge server. The resources available on each edge server is denoted by the resource vector $\langle$*RAM(MB), Cores, CPU Background Workload%, GPU Utilized%*$\rangle$. Since users are mobile and service requests are dynamic, the allocation algorithm discussed later in this section, is executed to obtain an allocation policy which decides the user-server binding whenever: (a) new users join the MEC environment, (b) users move away from the coverage of an edge server, (c) user service requests change, or (d) edge servers or mobile users go offline.

The goal of the allocation policy in this work is to serve as many possible service requests while strictly honouring the service execution latency threshold Γ. The knowledge of service execution time is necessary to make such decisions of whether to assign a user's request to an edge server. For deterministic allocation approaches, the execution time for services can be obtained from historical data by statistical methods, and then used to determine approximate values to obtain the allocation policy. However, due to the dynamic nature of execution time, the allocation policy can over or under allocate users to an edge server during a real execution scenario. In this work, we propose an RL based learning framework to obtain user-server binding decisions honouring the latency threshold γ by learning the service execution patterns from experience directly on the edge server.

The agent in the RL framework learns the environment to choose better action choices by exploring

the environment and receiving feedback from the action. The primary advantage of RL is to learn the underlying environment without requiring massive amount of labelled data. In this RL framework, the agent continuously interacts with the edge server to take actions i.e. execute several service requests and obtains corresponding rewards according to the execution footprint. As shown in Figure 5.4, the state is denoted by the resource vector of the edge server along with the number of service requests. The action set represents many service requests to be executed on the edge server. For our RL problem, the MDP [59] is represented as tuple $(\Sigma, A, R(a))$ as illustrated in Figure 5.5. The notations used throughout this chapter are shown in Table 5.2. The entries of the MDP are as follows:

- Σ is a finite set of states represented with six attributes as $\langle R$:RAM (in MB), C:Number of Cores, CW:CPU Background Workload%, GU:GPU Utilization%, N_{s1}:Number of service request for Service $s_1, \ldots, N_{sk}$:Number of service request for Service $s_k\rangle$. The values for RAM, Number of Cores, CPU Background Workload percentage and GPU Utilization percentage are taken from the resource vector of the edge server. Moreover, additional attributes are added i.e. the number of service requests for service $s \in S$. The number of service requests represents the number of users requesting to get served for the particular service hosted on the edge server. For example, the state $\langle 5000, 4, 40, 10, 100, 300\rangle$ represents an edge server that hosts two services s_1 and s_2 with the currently available resource of $5000MB$, 4 CPU cores, CPU background workload at 40% and GPU utilization of 10%, also, 100 users are requesting for service s_1 and 300 users are requesting for service s_2.

- A is the set of actions represented by the number of user requests executed on an edge server. An action $a_n \in A$ is represented by tuple $(p_{n1}, p_{n2} \ldots p_{nk})$, where k is the total number of services hosted on the edge server $e_n \in E$ and p_{nk} represents the number of service requests for service s_k to be executed. For example, for an edge server hosting two services s_1 and s_2, the action $(50, 100) \in A$ represents 50 requests for service s_1 and 100 service requests for service s_2 to be executed. Since the action space has size of $O(|U| \times |S|)$, we have reduced the cardinality of action space through quantization of size λ. For example, the quantization size $\lambda = 10$ produces a new action space where the new action tuple $(2, 2)$ represents all the actions in the range $(11 - 20, 11 - 20)$ in the old action space. We will discuss the impact of λ on performance in the experiments.

- $R(\sigma, a_n)$ is the immediate reward received after the agents take a particular action $a_n \in A$ from state $\sigma \in \Sigma$ on the edge server $e_n \in E$. The reward is computed from the service latency L due to an action a_n. Given a state $\sigma = \langle R, C, CW, GU, N_{s1}, \ldots, N_{sk}\rangle \in \Sigma$ and action $a_n = (p_{n1}, \ldots, p_{nk}) \in A$ on the edge server $e_n \in E$ hosting k services with hard service latency threshold of Γ, the reward is the sum of the services accomodated, multiplied by a damping factor η. Note that if the services chosen cannot be accomodated, then we have a reward of zero. Formally,

$$R_{lin}(\sigma, a_n) = \begin{cases} \eta \sum_{i=1}^{k} p_{nk} & \text{if latency } L < \Gamma \\ 0. & \text{Otherwise} \end{cases} \tag{5.1}$$

The agent in the MDP learns the optimal action i.e. the maximum number of service requests that can be deployed on the edge server so that latency L does not exceed Γ. The reward returned is high for actions with more number of service requests executed under a latency threshold of Γ, however, the reward is low whenever the service latency Γ is not honoured for certain actions. The agent learns the optimal action by exploring and exploiting the environment [59]. The environment for our problem is the real system which provides the real latencies (L) and state of the system. The latency generated by a particular action $a \in A$ is derived directly from the system by executing a number of services on

the edge server due to the action a. The service latency L is used to return the reward $R(a)$ for an action a.

Notations	Descriptions
U	The set of users $\{u_1, u_2 \dots u_i\}$
E	The set of edge servers $\{e_1, e_2 \dots e_j\}$
S	The set of services $\{s_1, s_2 \dots s_k\}$
Σ	State of the MDP $\langle R$: RAM, C: Cores, CW: CPU Background Workload(%), GW: GPU Utilization(%), N_{s1}: Number of requests for service s_1, ... N_{sk}: Number of requests for service $s_k\rangle$
A	Action space of the MDP
$R(\sigma, a_n)$	Reward due the action $a_n \in A$ at state $\sigma \in \Sigma$ on the edge server $e_n \in E$
λ	Quantization size for action space reduction
L	Service execution latency
Γ	The latency threshold
γ_{kj}	Latency of single request for service s_k at edge server e_j
η	Damping factor in reward function
α	Learning rate in Q-value update
β	Reward discount factor in Q-value update
U_j^{opt}	Predicted optimal number of service requests that can be provisioned on the edge server e_j
r_j	The coverage radius of edge server e_j
d_{ij}	The distance between user u_i and server e_j
$\Omega(u)$	Returns the index of service requested by user u

Table 5.2: List of Notations

The RL agent is trained using Deep-Q learning [39]. In Deep-Q learning, the states of the RL agent are input to a neural network and Q-values of each action are the outputs of the neural network. The Q-values at time step t for state s_t and action a_t are calculated according to the equation given in Equation 5.2 [59], where α is the learning rate and β is the discount factor. The optimal policy is to select the action with the maximum Q-Value. The agent can be trained on an edge server during service installations to predict the number of users that could get deployed on the edge server. This reduces the effort for offline training unlike the simple supervised learning approach.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{R(s_t, a_t) + \beta \max_{a_{t+1} \in A} [Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)\} \tag{5.2}$$

We show our heuristic to obtain the edge user-server allocation policy in Algorithm 5.1. The proposed algorithm performs a load balancing of service requests while computing the allocation policy. The optimal number of service requests U_j^{opt} that can be allocated to a particular edge server e_j predicted from the trained Deep Q agent is a tuple of sizes equal to the number of services hosted on the edge server i.e. $U_j^{opt} = (u_1^{opt} \dots u_k^{opt})$. For example, the tuple of predicted users $U_1^{opt} = (100, 200)$ with edge server hosting two services s_1 and s_2, signifies, the optimal number of service requests for service s_1

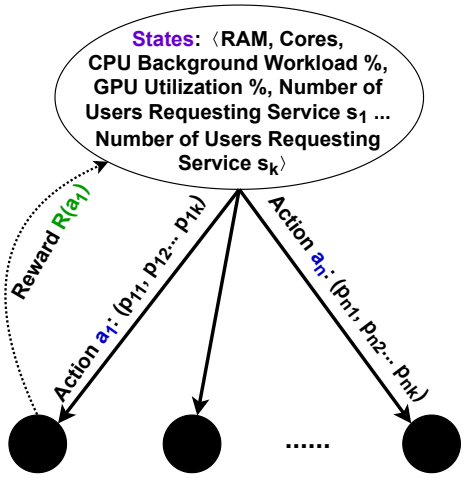


Figure 5.5: Reinforcement Learning Problem Model

is 100 and for service s_2 is 200 that can be accommodated on the edge server e_j while honouring the given latency threshold.

Algorithm 5.1: Algorithm for User Allocation with RL

Input : $U \leftarrow$ Users, $E \leftarrow$ Servers, DQN Agent
Output: Returns User-Server Allocation List $Alloc[]$

foreach $e \in E$ **do**
  $S \leftarrow$ State of the edge server e
  $U_e^{opt}[k] \leftarrow$ Tuple of predicted number of service requests given the state $\sigma \in \Sigma$ for an edge server e using the trained DQN RL Agent
end
foreach $u \in U$ **do**
  $e_{list} \leftarrow$ List of servers which cover user u
  $e_{selected} \leftarrow$ The server with maximum $U^{opt}[\Omega(u)]$ in the list e_{list}
  $Alloc[] \leftarrow$ Append $(u, e_{selected})$ which assigns user u to the server $e_{selected}$
  Decrement $U^{opt}_{e_{selected}}[\Omega(u)]$ by 1
end

5.4 Deterministic Approach Used as Baseline

Given the initial resource state vector on an edge server e_j, the execution time for a single request for service s_k is given by γ_{kj} on that particular edge server determined by averaging from historical service execution data i.e. deterministic execution time. The value of γ_{kj} will be used in our Integer Linear Programming[ILP] to determine the number of users that can be allocated to the edge server. The ILP formulation generates the user-server binding policy maximizing the number of users allocated to the edge servers.

The allocation of user $u_i \in U$ to the edge server $e_j \in E$ is denoted by the binary variable x_{ij} and the distance between the corresponding user and edge server is denoted as d_{ij}. The function $\Omega(u)$ returns the index of the service from S requested by the user $u \in U$. For allocation, the distance between the user and edge server d_{ij} should not exceed the coverage radius of the edge server r_j which is represented as a constraint in Equation 5.4. The total service execution latency caused due to users being assigned to a particular edge server e_j should not exceed the latency threshold Γ, as in

constraint Equation 5.5 for our ILP formulation. The constraint in Equation 5.6 ensures the allocation of a particular user to a maximum of only one edge server. The deterministic ILP allocation policy is developed along the approach followed in prior research works like [30] [31] etc. The formulation will be used to compare against our proposed approach with the RL agent.

Objective:

$$Maximize : \sum_{i=1}^{|U|} \sum_{j=1}^{|E|} x_{ij} \tag{5.3}$$

where,

$$x_{ij} = \begin{cases} 1, & \text{If user } u_i \text{ is allocated to server } e_j \\ 0, & \text{Otherwise} \end{cases}$$

Subject To:

1. Coverage Constraint:

$$d_{ij} \leq r_j \tag{5.4}$$

2. Latency Threshold Constraint:

$$\sum_{i=1}^{|U|} (x_{ij} \times \gamma_{\Omega(i)j}) \leq \Gamma \quad : \forall j \in \{1, \ldots |E|\} \tag{5.5}$$

3. User-Server Mapping:

$$\sum_{j=1}^{|E|} x_{ij} \leq 1 \quad : \forall i \in \{1, \ldots |U|\} \tag{5.6}$$

4. Integer Constraint:

$$x_{ij} \in \{0, 1\} \quad : \forall i \in \{1, ..|U|\}, \forall j \in \{1, ..|E|\} \tag{5.7}$$

The ILP formulation is NP-hard, so solving for the optimal allocation policy on a real system makes it difficult for ILP based allocation schemes to be implemented in the real world. Therefore, we also use a greedy heuristic based on the nearest neighbourhood allocation principle for comparison with RL based approaches, i.e., allocate the users to the nearest edge server with available resources to accommodate the user's request under the given latency threshold.

5.5 Experiments and Analysis of Results

All experiments are conducted on an Ubuntu machine with an Intel(R) Xeon(R) CPU E5-1650 v4 processor, 64GB RAM and Quadro P4000 8GB GPU. Two services, YOLO [52] and MobileNetV2 [56], are used widely used for object detection in images and videos. Both of these use CPU and GPU for computation, also, the applications are computationally heavy. So, the aforementioned services are used as representative services in our MEC environment to show the effectiveness of the learning approach to embracing more hidden parameters as GPU is also involved. All the programs are written in Python, the software library Stable-Baseline3 [50] is used for training of RL agents and the Python Mixed-Integer-Programming library [3] is used as the ILP solver. The results from our approach using

Parameter	Assumed Value
Initial RAM	3000 - 11000MB
Number of CPU cores	2 - 5
Initial CPU Workload	40 - 60%
Initial GPU Utilization	1 - 10%
Quantization Size λ	100
Damping Factor in reward η	0.01
Learning rate in neural network α	0.0001
Discount factor β	1
Number of time steps (episode)	5,00,000
Number of Users	100-500
Number of Servers	20-80
Latency threshold	20-50s

Table 5.3: Values of different parameters used in the evaluation

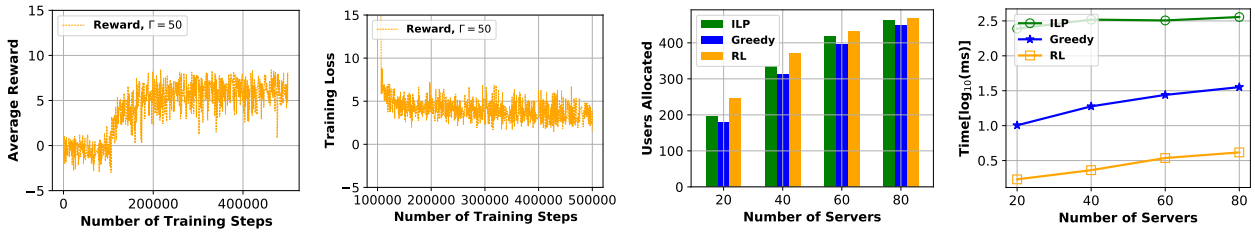
the RL agent discussed in Section 5.3 is compared with the deterministic formulation shown in Section 5.4 which is similar to the system modelling approach of [30] [31]. The number of users allocated and execution times for the algorithms are demonstrated for: (a) ILP in Section 5.4 **[ILP]**, (b) Greedy algorithm discussed in Section 5.4 **[Greedy]** and (c) Proposed RL approach with reward in Equation 5.1 **[RL]** to show the effectiveness of our work.

5.5.1 Experiment Setup

As done in our earlier chapters, we use the data-set for edge server locations as in [30], which includes data of base stations and users within the Melbourne Central Business District area. The coverage area of edge servers is set to 150 meters radius. The edge servers are assigned with the initial resource vector randomly as shown in Table 5.3. The RL agent proposed in this work is trained using the execution time obtained by executing YOLO and MobileNetV2 varying the RAM, Number of Cores, Workload (%), GPU Utilization (%) and Number of Service Requests. The trained model is then used for predicting the number of users that can be accommodated on the server given the state of the edge server. The quantization size of the action space is set to $\lambda = 100$. The damping factor in the reward function is set to $\eta = 0.01$ to avoid gradient overflow while training the DQN agent.

The DQN agent in the RL framework uses a neural network of 2 layers having 64 nodes at each layer with layer normalisation for our proposed model. The learning rate α is set to 0.0001 and discount factor β in Q-value calculation is set to 1. The exploration fraction for the agent is set to 0.4 for our experiment. The DQN agent is trained for 5,00,000 time steps with each step corresponding to one episode during exploration, which takes around 0.6 milliseconds for each time step during training of the agent. This took us an overall time of 50 minutes to train for 5,00,000 training steps. The γ_j for each server in the deterministic ILP is obtained from the server by executing YOLO and MobileNetV2, then computing the average time taken for execution given the initial resource state vector of the server.

Our experiment has by default a total of 500 users, the number of edge servers as 20-80, and latency threshold Γ of 50ms. We also have a set of experiments to study the influence of varying each of these individual parameters. We repeat each experiment 50 times and then compute the average allocation results for the sake of comparison. For each of the experiments, we show the (i) average reward using our technique, (ii) training loss using our technique, (iii) number of users allocated using our



(a) Average Reward (b) Training Loss (c) Allocated Users (d) Execution Time

Figure 5.6: Comparison of different performance parameters under the default configurations for our DRL scheme (RL) as well as the baseline techniques

technique and the baselines, and (iv) the execution time (in log scale) of running our technique and the baselines.

5.5.2 Result

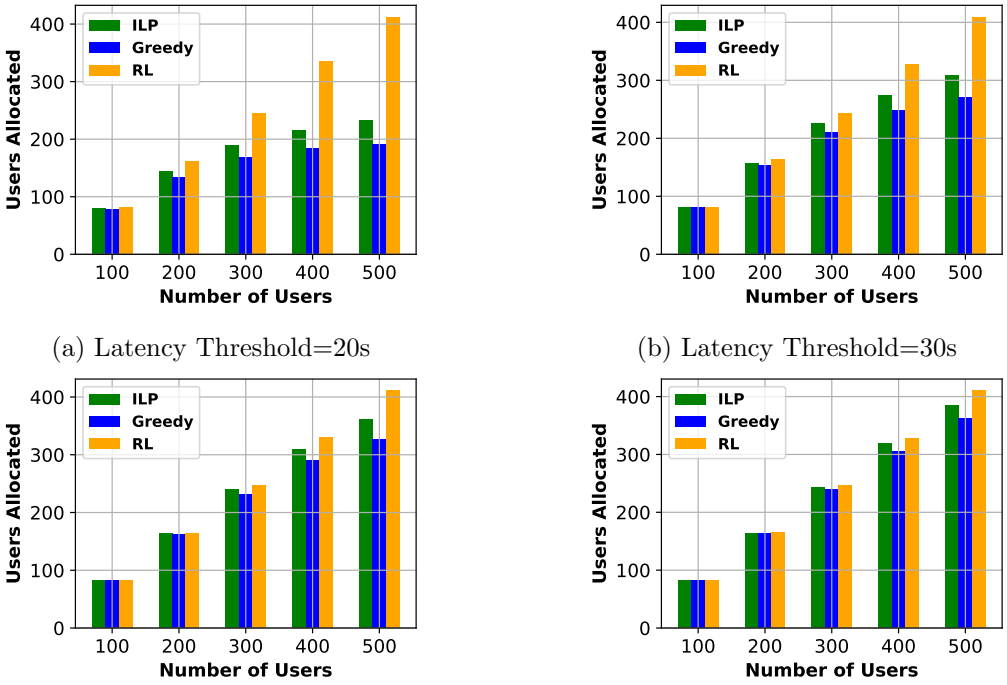
Default Configuration: Figure 5.6 shows the performance for the default configurations. We first note that the average reward converges at around 200,000 number of rounds. This corresponds to around 1200s of training time. After this, further training does not increase the reward significantly. Thus, the training loss also does not reduce from this point (Figure 5.6(b)).

We then compare the number of users that can be allocated for different number of servers from 20 to 80. We find that the number of users allocated is higher using RL as compared to ILP and Greedy by up to 16% and 18% respectively. This improvement is highest when the number of edge servers is equal to 40. On increasing the number of edge servers further, RL still performs better than ILP, but by a less amount. This is because when the number of servers is sufficiently high, even a simple allocation algorithm leads to allocation of most users. Thus, RL is most effective when the number of servers present is limited.

We also compare the performance in terms of execution time of each of these techniques. Once again, we find that RL has the least execution time, with allocation over even 80 servers taking only around 0.1s, in contrast to Greedy and ILP which take 0.32s and 1.2s respectively. Thus, RL performs better in terms of both number of users allocated as well as execution time under these configurations.

Performance under Varying Threshold Latency and Varying Users: We now consider the performance of RL and baseline techniques when we vary the threshold latency with number of users varying from 100-500 and the number of servers fixed at 50. Figure 5.7 shows the performance in terms of number of allocated users for each of the techniques. We find that the number of users allocated falls the least with a decrease in the threshold latency. This is different from the ILP and greedy techniques, where we find that there is a more significant decrease in performance with a reduction in the threshold latency. For example, when the number of users present is equal to 400, the ILP can allocate 280 and 310 for latency thresholds of 30s and 50s respectively. On the other hand, RL allocates around 320 users in both the cases. This shows the importance of RL especially in cases where the latency constraint is tighter. The execution time is lowest for RL outperforming ILP and greedy techniques.

Performance under Varying Threshold Latency and Varying Servers: We now consider the performance of RL and baseline techniques when we vary the threshold latency while varying the number of servers from 20-80 and number of users fixed at 500. Figure 5.8 shows the performance in terms of number of allocated users for each of the techniques.



(a) Latency Threshold=20s

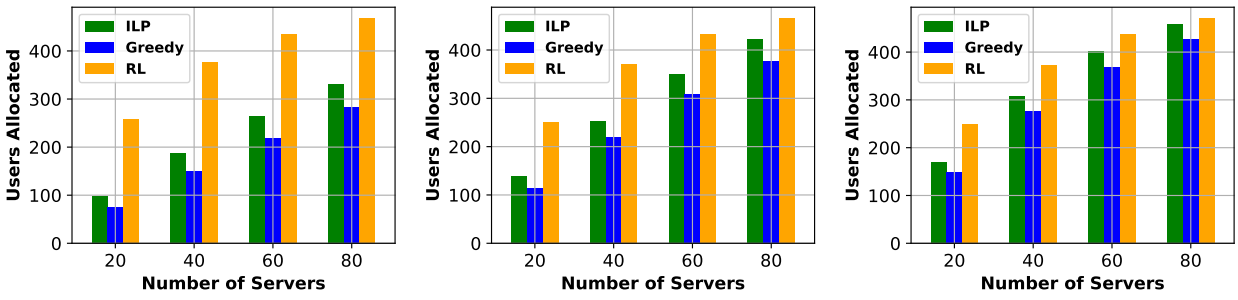
(b) Latency Threshold=30s

(c) Latency Threshold=40s

(d) Latency Threshold=50s

Figure 5.7: Comparison of the number of allocated users when the latency threshold is varied with varying number of users

We find that the difference in number of users allocated by RL from the other approaches to be significant when Latency Threshold is lower. The deterministic approaches struggle with allocated users with strict latency constraints. The number of servers in MEC affects the performance of the algorithm as more servers in the MEC environment makes the situation easy for the deterministic algorithm to allocate users comparable to RL. As expected, the execution time is less for RL than the deterministic approaches.



(a) Latency Threshold=20s

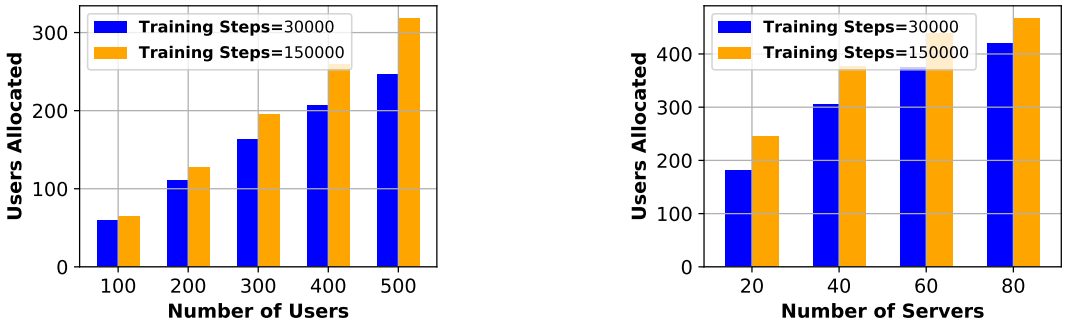
(b) Latency Threshold=30s

(c) Latency Threshold=40s

Figure 5.8: Comparison of the number of allocated users when the latency threshold is varied with varying number of servers

Impact of Training Time: We consider the number of allocations generated by an under trained RL agent with training steps $= 30,000$ and compare it against a properly trained RL agent with training steps $= 1,50,000$. The quantization size of action space is kept at $\lambda = 2$ for both the cases. As illustrated in Figure 5.9, Figure 5.9a shows the allocation with varying number of users with number of servers fixed at 30, while Figure 5.9b shows the result for varying servers with number of users fixed at 500 in the MEC environment with required Latency Threshold $\Gamma = 10s$. The trained

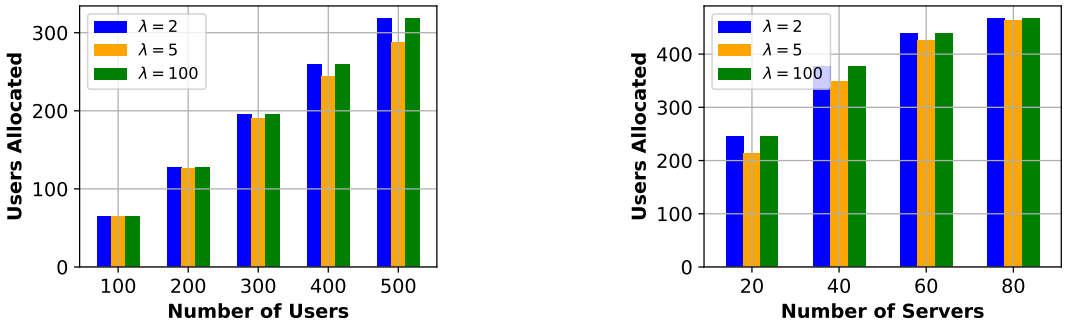
RL agent produces better results in comparison to the under trained agent as the under trained agent is inadequate to capture the non-linearity in execution time.



(a) Varying Users (b) Varying Servers

Figure 5.9: Impact of under training on allocation

Impact of Quantization Parameter: We consider the impact of quantization size λ on the RL agent for generating allocation policies. Figure 5.10 illustrates the effect of quantization size of $\lambda = 5$ with the quantization size of $\lambda = 2$. The model with $\lambda = 2$ produces better allocation results compared to a higher quantization size of $\lambda = 5$. The higher value of quantization size reduces the action space significantly while sacrificing the accuracy for the allocation, hence the allocation results with RL agent of quantization size $\lambda = 5$ performs poorly due the reduced accuracy of prediction.



(a) Varying Users (b) Varying Servers

Figure 5.10: Impact of Quantization parameter on allocation

5.6 Conclusion

In this chapter, we propose a DRL framework for online training and edge user allocation in the context of mobile edge computing. Our approach eliminates the need for modelling execution times, where we show using experiments that many of the standard assumptions related to resource utilization do not hold in practice. The proposed RL framework automatically infers resource utilization by executing services on the edge server and allocates users to edge servers while honouring the defined latency threshold. We perform experiments using real-world datasets and service execution data. Our experiments show that the RL based approach outperforms deterministic approaches that work based on historical execution footprint.

Chapter 6

Conclusion and Future Work

In this thesis, we primarily focus on the user allocation and service placements problems in the MEC context. Prior work in this area is predominantly oriented towards service provider centric allocations. Moreover, they fail to handle issues like workload fluctuations and non-linearity in the resource utilization of services on an edge server. In this work, we propose approaches to handle such issues by designing an efficient scalable user allocation and service placement policy.

To begin with, we propose a user-centric allocation approach that uses the QoS level preferences of users to design an allocation policy. We formulate an optimal ILP that considers user QoS level preferences to produce an optimal allocation policy. We propose a near-optimal heuristic algorithm based on the RB tree. We show that the number of users allocated is comparable with the ILP results with a simulated experiment on the real-world EUA dataset. Also, the time required to run our heuristic algorithm is significantly less compared to the ILP which makes it better for real-world scenarios.

Second, we rely on the effects of user QoS level preferences on the service resource utilization to model a resource variation aware allocation policy. We propose a stochastic model to handle workload fluctuations on the edge server. We formulate a deterministic integer programming model which jointly solves the user allocation and service placement problem. We perform experiments using the EUA dataset and show that the proposed approach fares better in allocating more users and efficiently utilizing resources on the edge server.

In our third contributory chapter, we move from modelling the MEC environment manually towards an automated learning based data-driven approach. We employ a DRL framework to implicitly learn and model the workload fluctuation and non-linearity resulting out of service executions directly on the edge devices. The learned DRL model is then used to obtain a user allocation policy honouring a defined latency threshold. We train the DRL framework using the real-world service execution footprint of two widely used services, YOLO and MobileNetV2. Next, we experimentally validate the efficiency of the proposed framework using the real-world EUA dataset. The results from the experiments are inspiring as it outperforms deterministic approaches by a significant margin.

In this era of growing IoT devices and Deep Learning-based applications, we believe this research adds a new direction to employ DRL based approaches for challenges in MEC. As future work, we plan to extend our findings to implement this concept on an actual edge testbed. Moreover, we plan to address the issues of mobility and dynamic service invocations with the RL approach. The DRL framework proposed in this work gets trained directly on the edge servers. We plan to explore the use of federated and transfer learning to minimize the training required for the DRL framework and make the DRL framework easily trainable on low compute devices.

Chapter 7

Disseminations out of this work

1. Panda S.P., Ray K. and Banerjee A., Dynamic Edge User Allocation with User Specified QoS Preferences. In proceedings of the 18th International Conference on Service Oriented Computing (ICSOC) 2020, Dubai, pages 187-197.

2. Panda S.P., Banerjee A. and Bhattacharya A., User Allocation in Mobile Edge Computing: A Deep Reinforcement Learning Approach. Accepted for Presentation at the 28th IEEE International Conference on Web Services (ICWS) 2021.

Bibliography

[1] ILOG CPLEX Optimization Studio - Overview.

[2] IoT Edge | Microsoft Azure.

[3] Mip: Python tools for Modeling and Solving Mixed-Integer Linear Programs (MIPs).

[4] Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. Mobile edge computing: A survey. *IEEE Internet Things J. 5*, 1 (2018), 450–465.

[5] Albanna, A. K., and Yousefi'zadeh, H. Congestion minimization of LTE networks: A deep learning approach. *IEEE/ACM Trans. Netw. 28*, 1 (2020), 347–359.

[6] Alfakih, T., Hassan, M. M., Gumaei, A., Savaglio, C., and Fortino, G. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. *IEEE Access 8* (2020), 54074–54084.

[7] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017* (2017), A. Akella and J. Howell, Eds., USENIX Association, pp. 469–482.

[8] Badri, H., Bahreini, T., Grosu, D., and Yang, K. Energy-aware application placement in mobile edge computing: A stochastic optimization approach. *IEEE Trans. Parallel Distributed Syst. 31*, 4 (2020), 909–922.

[9] Bertsekas, D. P., and Tsitsiklis, J. N. *Introduction to probability.* Athena Scientific, 2008.

[10] Bonomi, F., Milito, R. A., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC@SIGCOMM 2012, Helsinki, Finland, August 17, 2012* (2012), M. Gerla and D. Huang, Eds., ACM, pp. 13–16.

[11] Cai, Y., Yu, F. R., and Bu, S. Cloud computing meets mobile wireless communications in next generation cellular networks. *IEEE Netw. 28*, 6 (2014), 54–59.

[12] Chattopadhyay, S., and Banerjee, A. QSCAS: qos aware web service composition algorithms with stochastic parameters. In *IEEE International Conference on Web Services, ICWS 2016, San Francisco, CA, USA, June 27 - July 2, 2016* (2016), S. Reiff-Marganiec, Ed., IEEE Computer Society, pp. 388–395.

[13] Chattopadhyay, S., and Banerjee, A. A variation aware composition model for dynamic web service environments. In *Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings* (2018), C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds., vol. 11236 of *Lecture Notes in Computer Science*, Springer, pp. 694–713.

[14] Chen, J., Chen, S., Wang, Q., Cao, B., Feng, G., and Hu, J. iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks. *IEEE Internet Things J. 6*, 4 (2019), 7011–7024.

[15] Chen, L., Shen, C., Zhou, P., and Xu, J. Collaborative service placement for edge computing in dense small cell networks. *IEEE Trans. Mob. Comput. 20*, 2 (2021), 377–390.

[16] Chen, L., Zhou, S., and Xu, J. Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM Trans. Netw. 26*, 4 (2018), 1619–1632.

[17] Chen, W., Wang, D., and Li, K. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Trans. Serv. Comput. 12*, 5 (2019), 726–738.

[18] Chen, X. Decentralized computation offloading game for mobile cloud computing. *IEEE Trans. Parallel Distributed Syst. 26*, 4 (2015), 974–983.

[19] Chen, X., Jiao, L., Li, W., and Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw. 24*, 5 (2016), 2795–2808.

[20] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[21] Daga, H., Nicholson, P. K., Gavrilovska, A., and Lugones, D. Cartel: A system for collaborative transfer learning at the edge. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019* (2019), ACM, pp. 25–37.

[22] Farhadi, V., Mehmeti, F., He, T., Porta, T. L., Khamfroush, H., Wang, S., and Chan, K. S. Service placement and request scheduling for data-intensive applications in edge clouds. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019* (2019), IEEE, pp. 1279–1287.

[23] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *ASPLOS* (2019), pp. 3–18.

[24] Gan, Y., Zhang, Y., Hu, K., Cheng, D., He, Y., Pancholi, M., and Delimitrou, C. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019), pp. 19–33.

[25] Gao, Z., Jiao, Q., Xiao, K., Wang, Q., Mo, Z., and Yang, Y. Deep reinforcement learning based service migration strategy for edge computing. In *13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, San Francisco, CA, USA, April 4-9, 2019* (2019), IEEE.

[26] Guo, H., Liu, J., and Qin, H. Collaborative mobile edge computation offloading for iot over fiber-wireless networks. *IEEE Netw. 32*, 1 (2018), 66–71.

[27] Gupta, S., and Dinesh, D. A. Online adaptation models for resource usage prediction in cloud network. In *Twenty-third National Conference on Communications, NCC 2017, Chennai, India, March 2-4, 2017* (2017), IEEE, pp. 1–6.

[28] He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., Li, Y., and Yang, Y. A game-theoretical approach for user allocation in edge computing environment. *IEEE Trans. Parallel Distributed Syst. 31*, 3 (2020), 515–529.

[29] Hu, Y., Patel, M., Sabella, D., Sprecher, N., and Young, V. Mobile Edge Computing: A Key Technology towards 5G. Technical Report, ETSI, 2015.

[30] Lai, P., He, Q., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., and Yang, Y. Optimal edge user allocation in edge computing with variable sized vector bin packing. In *ICSOC* (2018), Springer, pp. 230–245.

[31] Lai, P., He, Q., Cui, G., Xia, X., Abdelrazek, M., Chen, F., Hosking, J. G., Grundy, J. C., and Yang, Y. Edge user allocation with dynamic quality of service. In *Service-Oriented Computing - 17th International Conference, ICSOC 2019, Toulouse, France, October 28-31, 2019, Proceedings* (2019), S. Yangui, I. B. Rodriguez, K. Drira, and Z. Tari, Eds., vol. 11895 of *Lecture Notes in Computer Science*, Springer, pp. 86–101.

[32] Li, J., Gao, H., Lv, T., and Lu, Y. Deep reinforcement learning based computation offloading and resource allocation for MEC. In *2018 IEEE Wireless Communications and Networking Conference, WCNC 2018, Barcelona, Spain, April 15-18, 2018* (2018), IEEE, pp. 1–6.

[33] Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y., and Kim, D. I. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Commun. Surv. Tutorials 21*, 4 (2019), 3133–3174.

[34] Ma, X., Zhao, Y., Zhang, L., Wang, H., and Peng, L. When mobile terminals meet the cloud: computation offloading as the bridge. *IEEE Netw. 27*, 5 (2013), 28–33.

[35] Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017* (2017), ACM, pp. 197–210.

[36] McClellan, M., Cervelló-Pastor, C., and Sallent, S. Deep learning at the mobile edge: Opportunities for 5g networks. *Applied Sciences 10*, 14 (2020).

[37] Minet, P., Renault, E., Khoufi, I., and Boumerdassi, S. Analyzing traces from a google data center. In *14th International Wireless Communications & Mobile Computing Conference, IWCMC 2018, Limassol, Cyprus, June 25-29, 2018* (2018), IEEE, pp. 1167–1172.

[38] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning, 2013.

[39] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (Feb. 2015), 529–533.

[40] Mohammadi, M., Al-Fuqaha, A. I., Guizani, M., and Oh, J. Semisupervised deep reinforcement learning in support of iot and smart city services. *IEEE Internet Things J. 5*, 2 (2018), 624–635.

[41] Nandi, B. B., Banerjee, A., Ghosh, S. C., and Banerjee, N. Stochastic VM multiplexing for datacenter consolidation. In *2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, June 24-29, 2012* (2012), L. E. Moser, M. Parashar, and P. C. K. Hung, Eds., IEEE Computer Society, pp. 114–121.

[42] Neto, J. L. D., Yu, S., Macedo, D. F., Nogueira, J. M. S., Langar, R., and Secci, S. ULOOF: A user level online offloading framework for mobile edge computing. *IEEE Trans. Mob. Comput. 17*, 11 (2018), 2660–2674.

[43] Ning, Z., Dong, P., Wang, X., Hu, X., Guo, L., Hu, B., Guo, Y., Qiu, T., and Kwok, R. Y. Mobile edge computing enabled 5g health monitoring for internet of medical things: A decentralized game theoretic approach. *IEEE J. Sel. Areas Commun. 39*, 2 (2021), 463–478.

[44] Ortiz, J., Lee, B., Balazinska, M., Gehrke, J., and Hellerstein, J. L. Slaorchestrator: Reducing the cost of performance slas for cloud data analytics. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018* (2018), H. S. Gunawi and B. Reed, Eds., USENIX Association, pp. 547–560.

[45] Ouyang, T., Li, R., Chen, X., Zhou, Z., and Tang, X. Adaptive user-managed service placement for mobile edge computing: An online learning approach. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019* (2019), IEEE, pp. 1468–1476.

[46] Pasteris, S., Wang, S., Herbster, M., and He, T. Service placement with provable guarantees in heterogeneous edge computing systems. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019* (2019), IEEE, pp. 514–522.

[47] Peng, Q., Xia, Y., Zeng, F., Lee, J., Wu, C., Luo, X., Zheng, W., Liu, H., Qin, Y., and Chen, P. Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019* (2019), E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds., IEEE, pp. 91–98.

[48] Poularakis, K., Llorca, J., Tulino, A. M., Taylor, I. J., and Tassiulas, L. Joint service placement and request routing in multi-cell mobile edge computing networks. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019* (2019), IEEE, pp. 10–18.

[49] Poularakis, K., Llorca, J., Tulino, A. M., Taylor, I. J., and Tassiulas, L. Service placement and request routing in MEC networks with storage, computation, and communication constraints. *IEEE/ACM Trans. Netw. 28*, 3 (2020), 1047–1060.

[50] Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[51] Ray, B. R., Chowdhury, M. U., and Atif, U. Is high performance computing (HPC) ready to handle big data? In *Future Network Systems and Security - Third International Conference, FNSS 2017, Gainesville, FL, USA, August 31 - September 2, 2017, Proceedings* (2017), R. Doss, S. Piramuthu, and W. Zhou, Eds., vol. 759 of *Communications in Computer and Information Science*, Springer, pp. 97–112.

[52] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), pp. 779–788.

[53] Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., and Kozuch, M. A. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (New York, NY, USA, 2012), SoCC '12, Association for Computing Machinery.

[54] Reiss, C., Wilkes, J., and Hellerstein, J. L. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, Nov. 2011. Revised 2014-11-17 for version 2.1. Posted at `https://github.com/google/cluster-data`.

[55] Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., and Wilkes, J. Autopilot: Workload autoscaling at google. In *Proceedings of the Fifteenth European Conference on Computer Systems* (New York, NY, USA, 2020), EuroSys '20, Association for Computing Machinery.

[56] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018* (2018), IEEE Computer Society, pp. 4510–4520.

[57] Schrijver, A. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[58] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal 3*, 5 (2016), 637–646.

[59] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[60] Tang, Z., Zhou, X., Zhang, F., Jia, W., and Zhao, W. Migration modeling and learning algorithms for containers in fog computing. *IEEE Trans. Serv. Comput. 12*, 5 (2019), 712–725.

[61] Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., and Wilkes, J. Borg: the Next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys'20)* (Heraklion, Greece, 2020), ACM.

[62] Trajano, A. F. R., de Sousa, A. A. M., Rodrigues, E. B., de Souza, J. N., de Castro Callado, A., and Coutinho, E. F. Leveraging mobile edge computing on smart grids using lte cellular networks. In *2019 IEEE Symposium on Computers and Communications (ISCC)* (2019), pp. 1–7.

[63] Wang, C., Liang, C., Yu, F. R., Chen, Q., and Tang, L. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE TWC 16*, 8 (2017), 4924–4938.

[64] Wang, F., Xu, J., Wang, X., and Cui, S. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. *IEEE Trans. Wirel. Commun. 17*, 3 (2018), 1784–1797.

[65] Wang, J., Hu, J., Min, G., Zhan, W., Ni, Q., and Georgalas, N. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Communications Magazine 57*, 5 (2019), 64–69.

[66] Wang, J., Zhao, L., Liu, J., and Kato, N. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Transactions on Emerging Topics in Computing* (2019), 1–1.

[67] Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., and Shen, X. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Trans. Mob. Comput. 20*, 3 (2021), 939–951.

[68] Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., and Shen, X. S. Delay-aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach. *IEEE Transactions on Mobile Computing* (2019), 1–1.

[69] Wang, S., Liu, H., Gomes, P. H., and Krishnamachari, B. Deep reinforcement learning for dynamic multichannel access in wireless networks. *IEEE Transactions on Cognitive Communications and Networking 4*, 2 (2018), 257–265.

[70] Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., and Leung, K. K. Dynamic service migration in mobile edge computing based on markov decision process. *IEEE/ACM Trans. Netw. 27*, 3 (2019), 1272–1288.

[71] Wood, L. 5G Optimization: Mobile Edge Computing, APIs, and Network Slicing 2019–2024. Tech. rep., Dublin, Ireland, Oct. 2019.

[72] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (New York, NY, USA, 2017), SoCC '17, Association for Computing Machinery, p. 452–465.

[73] Yin, Z., Yu, F. R., Bu, S., and Han, Z. Joint cloud and wireless networks operations in mobile cloud computing environments with telecom operator cloud. *IEEE Trans. Wirel. Commun. 14*, 7 (2015), 4020–4033.

[74] You, C., Huang, K., Chae, H., and Kim, B. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Trans. Wirel. Commun. 16*, 3 (2017), 1397–1411.

[75] Zappone, A., Debbah, M., and Altman, Z. Online energy-efficient power control in wireless networks by deep neural networks. In *19th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2018, Kalamata, Greece, June 25-28, 2018* (2018), IEEE, pp. 1–5.

[76] Zhang, W., Li, S., Liu, L., Jia, Z., Zhang, Y., and Raychaudhuri, D. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019* (2019), IEEE, pp. 1270–1278.

[77] Zhao, H., Deng, S., Zhang, C., Du, W., He, Q., and Yin, J. A mobility-aware cross-edge computation offloading framework for partitionable applications. In *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019* (2019), E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds., IEEE, pp. 193–200.

[78] Zhao, J., Li, Q., Gong, Y., and Zhang, K. Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Trans. Veh. Technol. 68*, 8 (2019), 7944–7956.

[79] Zhu, H., Cao, Y., Wei, X., Wang, W., Jiang, T., and Jin, S. Caching transient data for internet of things: A deep reinforcement learning approach. *IEEE Internet Things J. 6*, 2 (2019), 2074–2083.

[80] Zhu, R., Liu, B., Niu, D., Li, Z., and Zhao, H. V. Network latency estimation for personal devices: A matrix completion approach. *IEEE/ACM Trans. Netw. 25*, 2 (2017), 724–737.