



PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

Computation Offloading in Dynamic Mobile Environment

Submitted by

Gautham Bolar (PES1UG02EC044)
Sai Pranay Chennamsetti (PES1UG20EC048)
Dheemanth R Joshi (PES1UG20EC059)

January – May 2023 & August - December 2023

under the guidance of

Dr. Vamsi Krishna Tumuluru,
Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
PES University
Bengaluru-560085

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
PROGRAM B. TECH



CERTIFICATE

This is to certify that the Report entitled

Computation Offloading in Dynamic Mobile Environment

is a bonafide work carried out by

Gautham Bolar (PES1UG02EC044)
Sai Pranay Chennamsetti (PES1UG20EC048)
Dheemanth R Joshi (PES1UG20EC059)

In partial fulfillment for the completion of 8th semester course work in the Program of Study B.Tech in Electronics and Communication Engineering under rules and regulations of PES University, Bengaluru during the period Jan – Dec 2023. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements in respect of project work.

Signature with date

Name
Internal Guide

Signature with date & Seal

Dr. Shikha Tripathi
Chairperson

Signature with date & Seal

Dr. B. K. Keshavan
Dean -Faculty of Engg. and Technology

Name of the students

Gautham Bolar (PES1UG02EC044)
Sai Pranay Chennamsetti (PES1UG20EC048)
Dheemanth R Joshi (PES1UG20EC059)

Name / Signature of the Examiners:

Names

Signature

DECLARATION

We, **Gautham Bolar, Sai Pranay Chennamsetti, Dheemanth R Joshi**, hereby declare that the report entitled, *Computation Offloading in Dynamic Mobile Environment*, is an original work done by us under the guidance of **Dr. Vamsi Krishna Tumuluru, Professor, Department of ECE**, is being submitted in partial fulfillment of the requirements for completion of project work in the Program of Study B.Tech in Electronics and Communication Engineering.

PLACE: PES University, Bangalore - 85

DATE: 02/12/2023

NAME AND SIGNATURE OF THE CANDIDATES

1. Gautham Bolar :

2. Sai Pranay Chennamsetti :

3. Dheemanth R Joshi :

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to **Dr. Vamsi Krishna Tumuluru**, Professor, PES University for his exceptional guidance and unwavering support throughout this project. His expertise and mentorship have been instrumental in shaping the successful outcome of our endeavors.

We also extend our heartfelt thanks to **Dr. J Suryaprasad**, Vice-Chancellor, PES University for his support. We are also grateful to **Dr. Shikha Tripathi**, Chairperson, Department of Electronics and communication, PES University for her support in completing this project.

Finally, we thank our panel members, our parents, our friends and to all the people who were directly or indirectly involved, for their cooperation and support for guiding and supporting us in completion of this project work.

Gautham Bolar (PES1UG02EC044)
Sai Pranay Chennamsetti (PES1UG20EC048)
Dheemanth R Joshi (PES1UG20EC059)

Abstract

The demand for computing resources at the edge by vehicular users is rising rapidly. Resource allocation in dynamic mobile environment becomes significantly challenging. To address this, existing works use service migration and intelligent resource allocators to minimize service delay and maintain the Quality of Service (QoS). However, these works do not address joint communication and compute resource allocation in a service migration enabled environments. In this project, we try to jointly allocate compute and communication resources. We estimate vehicle pattern flow and model it as a random process. To represent the random process accurately, we formulate a Markov Decision Process (MDP). We also introduce network initiated service migration to follow the vehicle pattern. Finally, we use Deep Deterministic Policy Gradient (DDPG) based Deep Reinforcement Learning (DRL) algorithm to train a Deep Neural Network (DNN) based resource allocator and service migrator to minimize service delay and maintain consistent QoS. We provide extensive comparatives and performance metrics to demonstrate the efficiency of our model

Contents

Abstract	v
1 Introduction	1
1.1 Why Vehicular Edge Computing ?	1
1.2 Our Contribution and the Literature	2
1.3 Report Organization	4
2 Literature Review	5
2.1 Overview	5
3 System Model	9
3.1 Peripherals of the system model (Mobile Environment)	10
3.1.1 Vehicle	10
3.1.2 Road Side Unit (RSU)	10
3.1.3 Mobile Edge Computing Server (MEC Server)	12
3.1.4 Services	12
3.2 Service Migration	13
3.3 System Description	14
4 Formulation of Markov Decision Process (MDP)	17
5 DRL Trained Agent for Solving the MDP	20
5.1 Deep Deterministic Policy Gradient (DDPG)	20
5.2 Implementation of DDPG	21

<i>CONTENTS</i>	vii
5.3 Deployment of DRL model	21
6 Simulation and Results	24
6.0.1 Biasing model behaviour : Reward Modelling	26
6.0.2 A study on Migrations	28
7 Conclusion and Future Work	38
Appendix A	39
7.1 Components of a RL based system	39
7.1.1 Agent	39
7.1.2 Environment	39
7.1.3 Reward	40
7.2 History and State	40
7.2.1 Environment State	40
7.2.2 Agent State	41
7.3 Components of an RL agent	41
7.3.1 Policy	41
7.3.2 Value function	41
7.3.3 Model	41
7.3.4 Types of RL agents	42
7.4 Markov Decision Process	42
7.4.1 Markov Process	42
7.4.2 Markov Reward Process	43
7.4.3 Markov Decision Process	44
List of Figures	46
List of Tables	47

Chapter 1

Introduction

In recent years, the field of information technology has undergone a significant transformation with the rise of edge computing—a paradigm that emphasizes decentralized computation and data storage closer to the data source. This shift is driven by the need for real-time processing, especially in scenarios where low-latency responses are critical. Edge computing, by reducing the distance between data sources and computational resources, addresses the challenges posed by traditional cloud-centric architectures, making it particularly relevant for dynamic and time-sensitive environments.

1.1 Why Vehicular Edge Computing ?

As vehicles become equipped with advanced sensors, communication devices, and on-board computing capabilities, the volume and complexity of data generated within vehicular networks have increased substantially. Conventional centralized cloud computing models struggle to meet the stringent latency requirements imposed by safety-critical applications in vehicular environments [1]. Rapid decision-making, such as collision avoidance, navigation or traffic management, necessitates a computing infrastructure capable of processing data in near real-time.

There are several key motivations for vehicular edge computing:

1. **Low latency:** Vehicular edge computing can significantly reduce latency by processing data locally at the edge of the network, rather than transmitting it to a centralized cloud infrastructure. This reduces communication delays and improves overall network performance.

2. **Bandwidth optimization:** With the increasing popularity of Internet of Things (IoT) devices, the amount of data generated at the edge of the network is growing rapidly [2]. Vehicular edge computing can optimize network bandwidth by enabling data to be processed locally at the edge, reducing the amount of data that needs to be transmitted to a centralized cloud infrastructure.
3. **Improved reliability:** Vehicular edge computing can improve the reliability of connected and autonomous vehicles by enabling data processing to be done locally at the edge of the network, even in the event of network disruptions or outages.
4. **Enhanced privacy and security:** Vehicular edge computing can improve privacy and security by enabling sensitive data to be processed locally at the edge, rather than being transmitted to a remote cloud server. This is particularly important for applications that deal with sensitive data, such as healthcare, finance, and government services.

1.2 Our Contribution and the Literature

In existing literature, a significant amount of work has been done in the field of vehicular edge computing. The goal of this domain is to provide high Quality of Service (QoS) and keep the service delay to its minimum limit. Stability of these two parameters play a key role in providing application services like Augmented Reality (AR) and Virtual Reality (VR) and vehicle safety jobs. To achieve this however, is challenging.

Many challenges are involved in this system. First, the environment is highly mobile. This makes understanding of the traffic pattern difficult due to its random variations. From a vehicle's perspective, its trajectory traces various sections in the environment with varying network conditions, constant maintenance of QoS becomes challenging [1]. Another challenge associated with this scheme is the magnitude of service delay. With rapid increase in traffic volume in smart cities and highly latency sensitive applications and data processing at the edge, it becomes difficult to accommodate all the requests sent by the vehicles resulting in low latency service provision. Distance between service hosting MEC server and the vehicle also a sensitive parameter in determining the service delay. If this distance is majorly large, it will have significant impacts on the service delay of the

vehicle and may affect the QoS.

To address this issue, various schemes and algorithmic strategies have been implemented to solve the problem of dynamic mobile environment. Various studies mentioned in this project majorly dealt with intelligent resource allocation strategies to reduce the possibility of a service delay requirement. By allocating resources intelligently, the service provider tries to optimize various parameters to provide optimal service to the user in this sophisticated environment. To elaborate, the model deployed by the service provider allocates computing and networking resources to the user so that tasks can efficiently and collaboratively offloaded to the computing nodes present at the edge [3]. An optimization problem is formed which usually deals with the state variables which defines a parametric state of the environment and an action space which deals with various decisions the agent takes to maximize its profit. We found that Deep Reinforcement Learning (DRL) is a popular and impact making algorithm to solve the optimization problem.

To deal with the problem of tracking the vehicle and keeping the service close to the user as much as possible, Service Migration (SM) scheme is used to migrate services between the MEC servers to host the vehicle's service such that optimal delay will be faced [4]. We briefly describe and discuss about the concept of Vehicle Initiated Migration and Network Initiated Migration using [5] and [6] as our demonstrating examples. DRL is also used to take migration decisions in these scenarios.

We highlight the major drawbacks in the past implementations which exist in the literature. Majorly, we try to explore the possibility of vehicle initiated migration in a dynamic environment with unconstrained physical space.

Our contribution majorly lies in constructing and testing an offloading scheme which jointly allocates computing and communication resources to the vehicle subscribers such that we minimize the service delay to the subscriber. We leverage the scheme of service migration to migrate services to subsequent servers such that we minimize the service delay. We implement this through a network initiated migration based environment. We formulate a Markov Decision Process (MDP) to present the idea of the environment to the service provider. We use DRL model to learn the vehicle traffic pattern while trying to solve the MDP problem. Finally, we present detailed tests and performance analysis of this offloading scheme.

1.3 Report Organization

The problems discussed above have been individually mitigated using various methods in the existing literature, discussed in-depth in Chapter 2. Our system model, taking into consideration of the the pros and cons of the existing models is defined in Chapter 3.4. Formulation of Markov Decision Process (MDP) for our system model is given in Chapter 4. To solve MDP, we use Deep Reinforcement Learning algorithm explained in Chapter 5 followed by simulation in Chapter 6. The results and analysis of our model have been discussed in Chapter 6.

Chapter 2

Literature Review

2.1 Overview

The problem of task offloading in dynamic environments has been an active research area in the recent past. Existing literature addresses various problems and solutions to implement task offloading however, most of this work follows the same methodology. Traffic pattern in smart cities is significantly dynamic and random therefore, traffic patterns are modeled as a Random Process in the simulations to structure the data sensibly. With the evolution of *Deep Reinforcement Learning (DRL)*, which is becoming a popular method to learn the random processes, is being employed to address this issue. The problem is formulated as a *Markov Decision Process (MDP)*, which is further solved as an optimization problem using DRL algorithms. Therefore the focus now turns to modelling a robust environment which provides an ordered, well structured data which the DRL model can exploit. Most works around the literature revolve around environment modelling and data organization to learn the random process accurately.

One Direction of research focuses on centralized resource allocations. Essential network and compute resources such as Communication Bandwidth, Virtual cores of *Mobile Edge Computing (MEC)* server are allocated to the subscribers to the network, so that users can leverage computing resources present at the edge to process data and compute intensive applications. Authors in [7] proposed a multidimensional resource allocation approach. They also exploit the power of DRL

to learn the network environment parameters. *Deep Deterministic Policy Gradient (DDPG)* was used to train the network in offline mode to learn the continuous action space in the environment. [8] proposed a vehicular edge network architecture in which *User Entities(UE)* as a service was proposed. Vehicles in the network were considered as serving edge nodes to process network data. Again, DRL model was trained to learn the optimal policy for resource allocations for network subscribers.

Another path is to opt decentralized approach. [9] proposed a decentralized approach to take task offloading decision. All the subscribers in the environment have the liberty to offload the task into the network. The delay is estimated and offloading is optimized using a moving average model. This work did not employ DRL, however provided promising offloading strategy. However, this paper made assumption that the subscribers are static in the environment.

To keep up with the dynamic variations in the environment, various methods were proposed in the literature. One primary method to keep computing resources close to the subscriber is Service Migration. Computing nodes present at the MEC server are structured as virtual computing cores and are assigned to subscribers present in the network. Further, vehicle takes the decision to offload compute intensive tasks to the MEC server. Service Migration is one of the most active research scheme for vehicular edge computing scenarios.

Several methods on Service Migration has been studied and covered in this survey. Authors in [5] jointly optimize routing and service migration decisions to reduce service delay for all the vehicles. Further, the service and load pattern is learnt by each vehicle by using DRL model. [10] proposed a single agent problem in which the simple scenario of service migration between two MEC servers was discussed. Authors used real time data to formulate their MDP. Further used DRL to solve the MDP. work in [11] framed a scheme to handle the process of service migration efficiently. In particular, authors demonstrated methods like pre-copy/post copy to efficiently migrate services to the successor server. [12] have proposed an "online" model training approach to adapt to the real time pattern of the traffic, through which they achieve consistent Quality of Service (QoS).

Authors in [6] paper consider a closed pedestrian environment to group the subscribers based on various parameters to provide service accordingly. Therefore, grouping the users into user groups

gives an idea of the attribute locality of the user, through which high quality and identical services can be provided to large group of users.

With this detailed study of the literature, we now focus on two major implementations of service migration. We primarily classify them into "Network Initiated Migration" and "Vehicle Initiated Migration". To demonstrate the above processes, we take [6], [5] as our examples.

[5] employs "Vehicle Initiated Migration", which simply implies that the service migration decision is taken by the "Vehicle" and in most cases acts as the MDP agent. In this case, the vehicle employs a greedy approach, through which it focuses to maximize only its own profits and makes decisions accordingly. Though this model succeeds in maximizing local rewards, the affect of an arbitrary vehicle's decision significantly affects the other, in terms of QoS and service delay. The paper also assumes that the communication bandwidth between the Road Side Unit (RSU) and the vehicles present in its vicinity will be allocated equally, which can lead to severe communication bottlenecks for large data applications. The paper jointly optimizes routing and service migration decisions, which binds the vehicle to follow the routing policy provided by the agent, which if failed to execute, can cause severe QoS discontinuities.

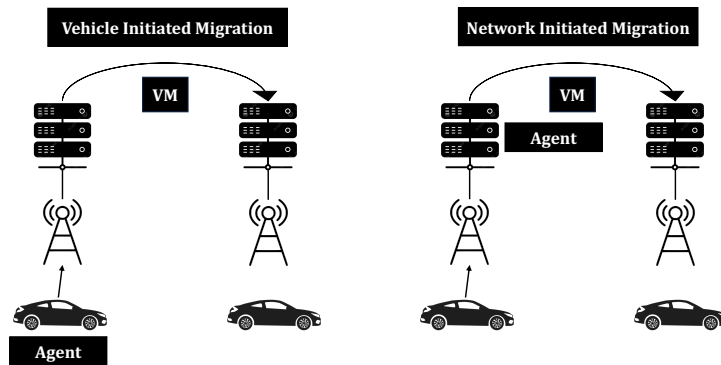


Figure 2.1: Left: Vehicle Initiated Migration, Right: Network Initiated Migration

Paper [6] approaches the problem through "Network initiated Migration", through which the agent can obtain global scope of the environment through the network and allocate resources to the

subscribers as a global observer. The model proposed in this work groups the users using standard clustering algorithms using standard parameters such as position and velocity. However, this implementation assumed that there are constant number of vehicles moving around the environment. Assumptions also include equal bandwidth allocations for all the clusters.

Chapter 3

System Model

As discussed in 2, there are a few conditions which were not discussed in the existing literature. For example, in [6], [5] authors have decided to divide the bandwidth equally. If a subscriber has a large data to compute with a stringent delay tolerance, QoS can be compromised due to the communication bandwidth between the subscriber and the network becomes the bottleneck. Another issue to be addressed in [5] is the concept of vehicle initiated migration. In a vehicle initiated migration scenario, the vehicle has no information regarding the network load which might lead to inefficient selections of computing resources. In the case of Network initiated migration, which is implemented in [6], the network collects impacting data from the subscribers and aims to optimize resource allocations for all the subscribers keeping a view of the global scope.

In this chapter, we will be introducing our system to approach this problem. First, we will be introducing various subsystems involved in the system model. Next, we will be putting all the subsystems to make up the environment of simulation.

In essence, our system model is a simulation environment. Moving cars handing off their corresponding tasks to the MEC server will be simulated. We will need a variety of computing and communication peripherals to begin with; these will be covered in the section that follows. Our goal is to efficiently distribute computing and communication resources such that the majority of users using our network's service do not experience a deficiency in quality of service. Network Initiated Service Migration is another tool we use to adapt to our changing environment. This will be accomplished with the use of a DRL model. To get an ideal picture of the status of the environment, we
















	Bandwidth Resource Allocation	Compute Resource Allocation	User Grouping?	Vehicle Initiated Migration?	Network Initiated Migration?
[1] Q. Yuan <i>et al</i>					
[2] W.Chen <i>et al</i>					
Our Approach					

Figure 3.1: Comparison of our work with [6] and [5]

will make use of the DRL model's capacity to learn random processes in noisy environments.

3.1 Peripherals of the system model (Mobile Environment)

3.1.1 Vehicle

Vehicle instances are deployed in the environment. Vehicles act as our subscribers and intend to leverage our service by demanding computing resources at our servers. Vehicles are assumed to offload data and request computing resources to the network randomly.

Vehicle Model Description

Vehicles are injected into the environment using a Poisson process. They maintain a random velocity sampled from a truncated Gaussian distribution. More details on this is provided in Chapter Simulation.

3.1.2 Road Side Unit (RSU)

To collect collect data from the vehicles through a wireless medium and submit it to the MEC server, we will be using a high speed 5G base station called a Road Side Unit (RSU). RSUs establish

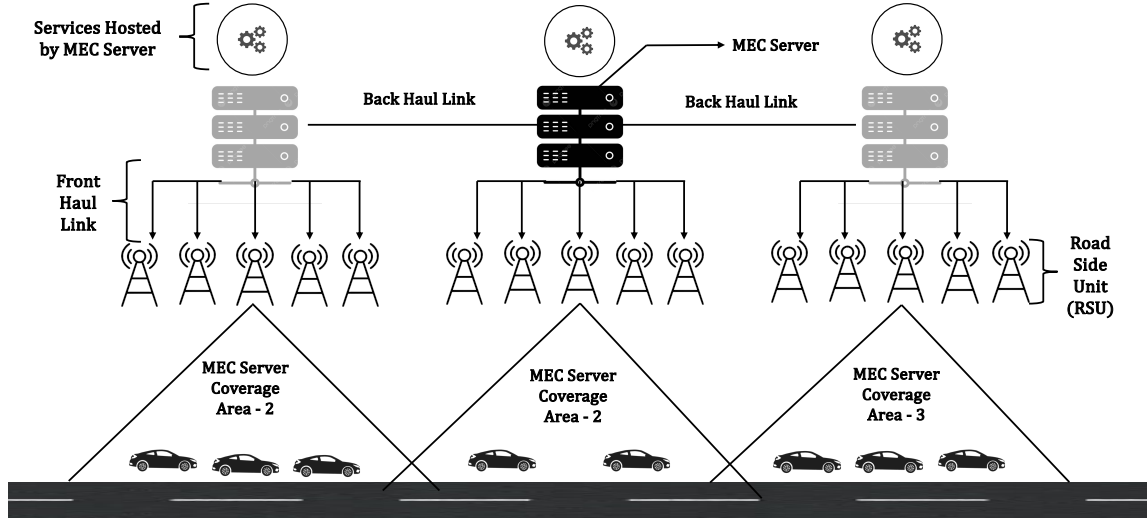


Figure 3.2: System Model Overview

connection between the vehicles through a wireless medium. RSUs allocate bandwidth resources to the vehicles which enable the vehicles to offload the data to the edge server. Multiple RSUs are connected to the edge servers through the wired high speed front-haul link.

RSU Model Description

RSUs are simulated as entities which will be assigned a physical position in the environment. Based on the position of the vehicle with respect to the RSU and bandwidth allocated to the vehicle, upload speeds for each vehicle can be calculated. RSU also provides information about the state of the environment. This state information is accumulated with respect to all RSUs and is sent to the MEC server.

Bandwidth Resource Blocks (BRBs)

To model the RSU's Bandwidth resources accurately, we introduce the term called Bandwidth Resource Blocks (BRBs). The BRBs are allocated to the subscribers in the environment. The total bandwidth spectrum in which the RSU is operated is divided into equal frequency ranges, and maintained in the RSU. Based on the network's decision, these BRBs are allotted to the subscribers based on their requirement. Therefore, users having large amounts of data to be offloaded to the network

will be allotted comparatively more amount of BRBs when compared to a subscriber having less amount of offload data. Therefore, we bring a freedom of bandwidth adaptability into the system to maintain consistent QoS even in dynamic environment.

3.1.3 Mobile Edge Computing Server (MEC Server)

The intent and the data sent by the vehicles is sent to a MEC server via the RSUs. In our implementation, we assume that multiple RSUs will report to a single MEC server. The server contains computing units (compute cores) which is treated as a resource to be allocated to the server. The computing resources hosts a vehicle group's Virtual Machine (VM) and a set of virtual cores of the server are allocated to the vehicle group. The vehicle group runs its desired applications on the MEC server. Vehicle grouping methodology will be discussed in the next section.

Compute Resource Blocks (CRBs)

To host a virtual machine and run applications of the users, we are required to allocate compute cores to the users. Compute cores in our scenario are modeled as virtual cores i.e we assume that we will have a compute capacity of F (GHz) and this frequency can be allotted to the users based on requirement. To simplify the resource allotment, we will divide our compute capacity into Compute Resource Blocks (CRBs). CRBs like BRBs, will be allocated by the network based on the computing intensity and data offloaded by the user.

3.1.4 Services

We provide "Compute as a Service" to our subscribers. Our services are modelled as data and compute demanding applications running at the edge of the network. To successfully implement this, we model our users's job as an "Application". Applications are executables hosted at the edge are usually related to Augmented Reality (AR), Virtual Reality (VR) applications which usually are deployed to provide In Vehicle Infotainment (IVI) to the users. However, running these applications at the edge is challenging in many ways, some of them include compute resource requirement and latency sensitive jobs. These applications are further run in the hosted VM of the MEC server.

To summarise and connect the dots mentioned in the concepts of this section, We will be modelling

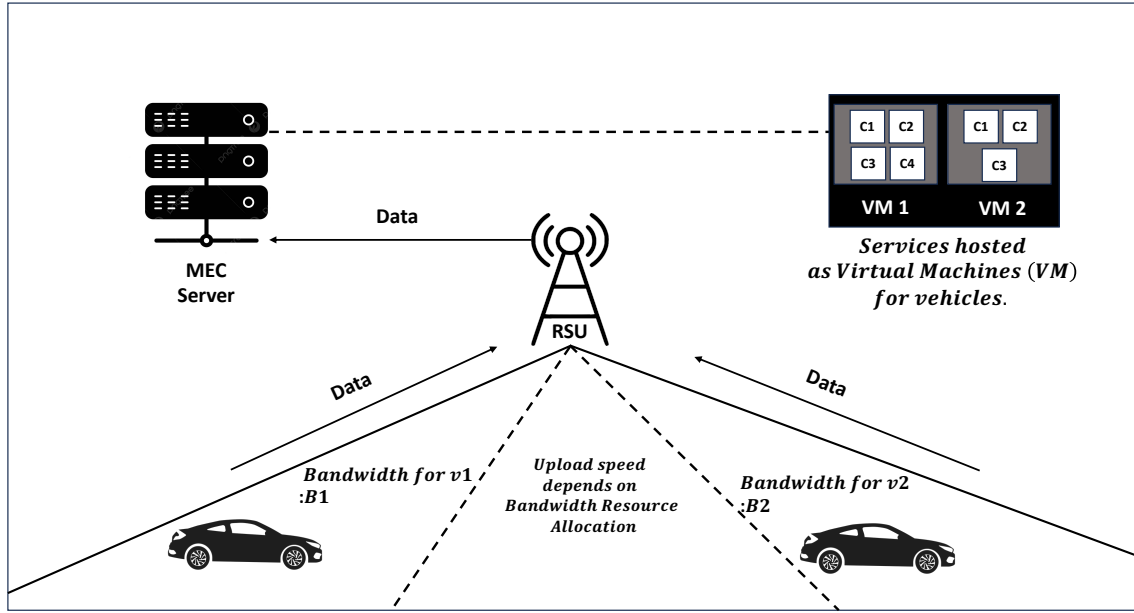


Figure 3.3: Workflow of the system

a dynamic mobile environment and allocate optimal resources to the subscribers so that they face minimum service delay and the QoS remains stable. The edge network hosts a set of applications which can be used by the users. Users offload data and compute intensive tasks to the MEC server through the RSU. Offloaded data is computed via an application hosted on the vehicle group's VM. The computed data is then sent back to the users.

3.2 Service Migration

As discussed in previous sections, we are dealing with a mobile environment. The environment's dynamic and mobile behaviour makes resource allocation and QoS management challenging. To address this issue, we will be using the concept of service migration. The process of migrating a service from initial serving node to next service node is called service migration. Since the vehicles are displaced constantly, service delay might severely increase if the service was hosted on the same server until vehicle is under the network. Therefore, vehicle's service is migrated to successive serving edge nodes. The intent behind this method is to keep the distance between the vehicle and the edge server hosting server minimum. To achieve this, the vehicle's allotted VM is migrated to the successive serving edge node. Fig. 3.4 demonstrates service migration

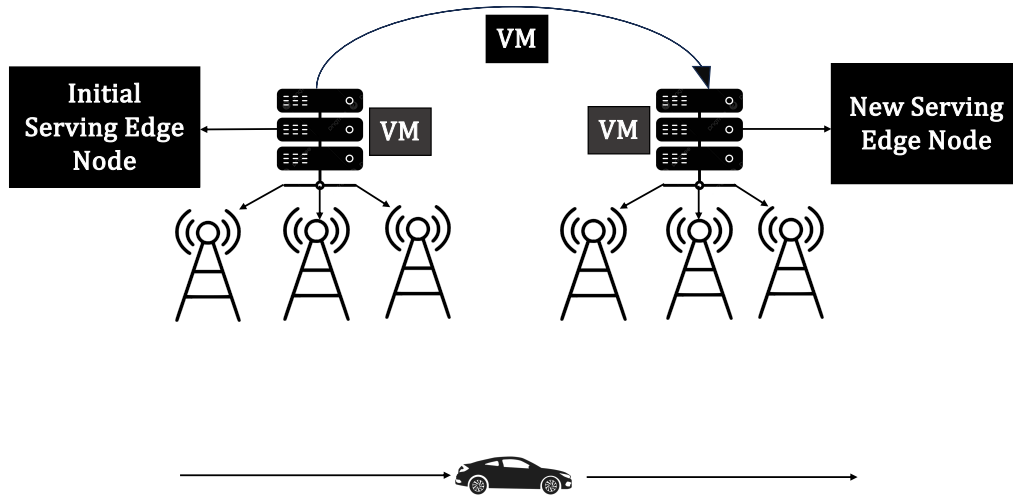


Figure 3.4: Service Migration

3.3 System Description

We now will build our model architecture. To start, we will build a road network. For this project, we will be assuming a simple three segment road network with a unidirectional traffic flow. Each segment is installed with five RSUs and one MEC server. All RSUs present in the environment is connected to the MEC server via front-haul link as discussed in previous section. MEC servers maintain their own network to communicate with each other, at their hierarchical level. to enable this, we connect each server with a back haul link. Fig. 3.2 briefly describes our system architecture.

As demonstrated, the vehicles can offload the tasks to the edge server as per application requirement. It can be seen that the data offloaded by the vehicle is collected by the in-coverage RSU and will be delivered to the edge server for computation.

The MEC server will perform three tasks operations based on the data and other parameters received by the vehicles.

1. User Grouping based on Application type, velocity and serving RSU, as observed in the environment.
2. Decide upon Communication and Computing resources allotment for user groups for the next

time step.

3. Take service migration decision for a vehicle group and migrate the service based on the decision.

More on User Grouping is discussed in the next chapter.

Based on the scheme described above, we will be formulating a single agent based Markov Decision Process (MDP) to describe the impact of an action taken by the network for a given state of the environment. To obtain the quantified values of the state variables, we observe the traffic patterns and intensities in our environment. As discussed in previous sections, the traffic intensity and traffic flow in the environment varies randomly with time. However, to simulate the scenario efficiently, we model the traffic parameter variations as a Random Process. The Random Process formulation for our environment is discussed in Chapter 6. To learn the random process efficiently, we make use of a DRL model. The DRL model acts as the MDP "Agent" to take actions in the environment and learns the value of its action through positive or negative reinforcement exposed to it through a reward function.

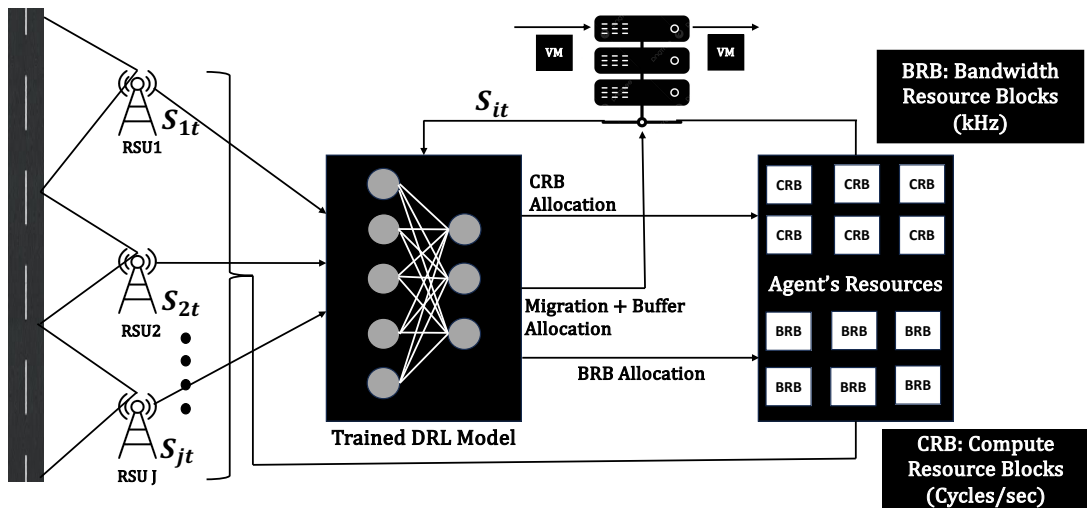


Figure 3.5: System Integrated with DRL Model

In the next chapter, we will be describing our environment structure. We will be modelling our

MDP and demonstrate our state and action vector. Table.3.1 provides reference for notations and their meaning.

Table 3.1: Notation and Brief Description Table

Notation	Brief Description
V	Set of vehicles $v \in V$.
\mathcal{J}_i	Set of RSUs managed by the edge server i .
\mathcal{K}	Set of services denoted by their ID $k \in \mathcal{K}$.
\mathcal{T}	Dwell time limit.
X_i	Random Variable denoting the number of vehicles migrated by server i to server $i + 1$.
Y_i	Random Variable denoting number of CRBs reserved by server i for the migrations of server $i - 1$.
C_k	Number of compute cycles required by service k .
L_k	Size of the data offloaded by service k .
τ	Service delay threshold.
τ_{\max}	Maximum service delay threshold.
B	Size of a bandwidth resource block.
$b_{j,k}$	BRBs allocated by RSU j for the user group running service k .
c_k	CRBs allocated by the edge server for the user group running service k .

Chapter 4

Formulation of Markov Decision Process (MDP)

In mathematics, a Markov decision process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. We begin our formulation of the optimization problem as the following MDP.

Let \mathcal{J}_i represent the set of RSUs which are managed by the edge server i . A vehicle can be served by the edge server i as long as it remains in the coverage of any RSU $j : j \in \mathcal{J}_i$, although its service may be hosted in edge server $i + 1$.

The State S_t observed by the edge server at time index t is defined as the vector $S_t = [\{V_{j,k,t} : \forall j, \forall k\}, W_t, X_{i-1,t}, Y_{i+1,t}]$ where $j \in \mathcal{J}_i, k \in \mathcal{K}$. W_t is the number of vehicles with remaining time less than \mathcal{T} under server i 's coverage, these vehicles are considered for migration by the edge server. The action vector, denoted by a_t is defined as $a_t = [\{b_{j,k,t} : \forall j, \forall k\}, \{c_{k,t} : \forall k\}, y_{i,t}, x_{i,t}]$. The process does not have a state transition equation nor does it have an underlying distribution $P(S_{t+1} | S_t, a_t)$ from which we sample the next state S_{t+1} given S_t and a_t . Hence the given MDP is termed as being “model-free”.

A Policy π is a mapping from states to actions. A policy may be deterministic (i.e $\pi(s)$ or stochastic (i.e $\pi(a | s)$).

The Reward emitted by the process for a given pair of $\langle s_t, a_t \rangle$ is denoted by $R_t(s_t, a_t)$ which is defined as,

$$R_t(s_t, a_t) = \sum_{\forall j} R_{j,t}(s_t, a_t) \quad (4.1)$$

$$= \sum_{\forall j} \sum_{v \in V_j} U_{v,t} \quad (4.2)$$

where $U_{v,t}$ is termed as the utility function defined as,

$$U_{v,t} = \begin{cases} 1 & \text{if } D_{v,t} < \tau \\ \frac{\tau_{\max} - D_{v,t}}{\tau_{\max} - \tau} & \text{if } \tau < D_{v,t} < \tau_{\max} \\ 0 & \text{if } D_{v,t} > \tau_{\max} \end{cases} \quad (4.3)$$

where $D_{v,t}$ is the service delay encountered by vehicle v at time t . Suppose vehicle v is running service k . Then $D_{v,t}$ is given by,

$$D_{v,t} = D_{v,t}^{\text{tr}} + D_{v,t}^{\text{co}} \quad (4.4)$$

$$D_{v,t}^{\text{tr}} = \frac{L_k}{(b_{j,k,t}/|V_{j,k,t}|)B\log_2(1 + \text{SNR}_{v,t})} \quad (4.5)$$

$$D_{v,t}^{\text{co}} = \frac{C_k}{c_{k,t}/(\sum_{j \in \mathcal{J}_i} |V_{j,k,t}| - x_{i+1,k,t})} \quad (4.6)$$

where $D_{v,t}^{\text{tr}}$ and $D_{v,t}^{\text{co}}$ are the transmission and compute delays respectively. The utility function mentioned in eq.(4.3) is a measure of the perceived QoS by the user. SNR_v is the signal to noise ratio between the the RSU and vehicle. It is given by,

$$\text{SNR}_v = \frac{Ph(d/d_o)^{\alpha(\epsilon-1)}}{N} \quad (4.7)$$

Where P is the transmit power of the vehicle, h is the channel gain, d is the distance between the vehicle and RSU, d_o is a reference distance, α is the path-loss exponent, ϵ is the power control factor and N is the noise power.

With these pieces in place, we may now discuss about the optimization objective of our problem. The goal in a MDP is to find the optimal policy π^* which will maximize the expected cumulative function of rewards i.e,

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^T \gamma^t R_t(s_t, a_t) \right] \quad (4.8)$$

where $\gamma \in (0, 1]$ is called the “Discount factor”. By substituting eq.(4.2) into the above equation and defining certain constraints on the actions, we obtain the equation of our optimization problem,

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^T \sum_{\forall j} \sum_{v \in V_j} \gamma^t U_{v,t} \right] \quad (4.9)$$

subject to the constraints,

$$0 \leq \sum_{\forall k} b_{j,k,t} \leq \bar{B} \quad \forall j \quad (4.10)$$

$$0 \leq \sum_{\forall k} c_{k,t} + y_{i,t} \leq \bar{C} \quad (4.11)$$

$$0 \leq x_{i,t} \leq W_t \quad (4.12)$$

eq.(4.10) sets a limit on the total allocations per RSU to be under its total bandwidth resources. eq.(4.11) sets a limit on the total compute resources that can be allocated for the service groups and migrations from server $i - 1$ to be under the server’s total compute capacity. eq.(4.12) sets the maximum possible migrations to be less than or equal to the number of users who have a remaining time less than \mathcal{T} under the server’s coverage.

We may now discuss about the procedure for finding the optimal policy π^* .

Chapter 5

DRL Trained Agent for Solving the MDP

In this chapter, we will be discussing regarding training and Deploying our DRL model. Using the data organisation shown in previous chapters which formulated as a Markov Decision Process (MDP), we will be constructing and training a Deep Neural Network (DNN) using Reinforcement Learning. First, we will look into Deep Deterministic Policy Gradient Algorithm, which is an RL based training method for models dealing with continuous action spaces. Next, we will look at our neural network architecture and training flow to learn the optimal policy π^* . Finally, we see the deployment of our agent in our environment to measure its performance.

5.1 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an algorithm which learns both the Q value function and optimal policy π^* concurrently.

The algorithm's essence lies in Q-learning and learning the optimal policy by using the Bellman Equation.

Usually, Q-learning is modelled as a look up table, in which each state action pair maps to a q value which is learnt and updated by exploring and exploiting the environment multiple times. However, this method becomes complex and time consuming in large and continuous action spaces. To deal

with such situations DDPG is used where instead of looking them up as an input-output mapping, we try to approximate them. To achieve the approximation, we use Deep Neural Networks (DNNs) which hold the title of Universal Function Approximators. This method significantly reduces the memory and time complexity. the DDPG involves four DNNs to train the agent. The four networks are Target actor and critic, and Behavioural actor and critic. The main idea is to learn the policy by Q-learning, where the actor gives an action for each state as the input. The critic approximates the value of each action given by the actor. For the behavioural actor critic, the gradients are updated as shown in steps 13,14 in Algorithm 1 respectively. The target actor critics are gradually smoothened and regressed to the behavioural actor and critic. Eventually, the target networks can accurately output the desired decisions which in our case compute and bandwidth allocations, and service migration decisions respectively. DDPG Alogrithm is demonstrated in Algorithm 1.

5.2 Implementation of DDPG

As seen in previous section, DDPG holds the ability to learn the continuous action spaces which is an important requirement in our scenario. In this section, we will be constructing our agent neural network following Algorithm 1. We set our neural network parameters as shown in Table.

Table 5.1: Network Parameters (Actor)

Parameter	Value
Number of Hidden Layers	1
Input: State Vector Size	18
Output: Action Vector size	20
Hidden Layer size	100
Activation Function	tanh, sigmoid

5.3 Deployment of DRL model

Once the target actor is trained with satisfactory value evaluation by the critic, we deploy the DRL model in our simulation environment. This is demonstrated in Fig. 3.5.

The DRL model is expected to make computing and bandwidth resources to the vehicle sub-

Algorithm 1 Deep Deterministic Policy Gradient

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\theta_{\text{targ}} \leftarrow \theta$ ,  $\phi_{\text{targ}} \leftarrow \phi$ 
3: repeat
4:   Observe state  $s$  and select action  $a = \text{clip}(\mu_{\theta}(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for however many updates do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute targets
          
$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

13:      Update Q-function by one step of gradient descent using
          
$$\nabla_{\phi} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi}(s, a) - y(r, s', d))^2$$

14:      Update policy by one step of gradient ascent using
          
$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi}(s, \mu_{\theta}(s))$$

15:      Update target networks with
          
$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho) \phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$

16:    end for
17:  end if
18: until convergence

```

Figure 5.1: DDPG Algorithm

scribers at each time instant. This aims to optimize the utility function, which mainly tries to maintain the QoS. The agent also decides to migrate the services to its subsequent MEC server optimally.

Table 5.2: Network Parameters (Critic)

Parameter	Value
Number of Hidden Layers	1
Input: Vector Size	38
Output: Action Value size	1
Hidden Layer size	100
Activation Function	tanh, ReLU

Chapter 6

Simulation and Results

The formulation given above was used to simulate resource allocations and migrations in an edge server overseeing a road with unidirectional traffic flow. The simulation is run for a duration of one hour. Simulation parameters which remain constant throughout all runs are specified in table.6.1. Parameters which change during runs are specified explicitly.

Vehicles are injected into the simulation according to a Poisson process λ_v , migrations are made according to λ_m and allocations by server $i + 1$ are made according to λ_a . Vehicle velocities are sampled from a truncated Normal distribution $\mathcal{N} \sim (\mu_v, \sigma_v^2)$ with limits v_l and v_u . The simulation's time scale is decided by the fastest moving vehicle i.e v_u . The time scale is equal to the time taken to traverse through the coverage of a single RSU, which gives us a value of 1.5 minutes.

A base time series is generated by running the simulation for the specified duration of one hour. This series can be thought of as a prediction/forecast of the hourly traffic, migrations and allocations obtained by some prediction method/network. The forecast will be subject to prediction errors. Let X denote the forecast quantity and \hat{X} denote the actual value of the quantity.

$$\hat{X} = X + \delta \cdot X \tag{6.1}$$

where $\delta \sim \mathcal{N}(0, \sigma^2)$ and $\delta \cdot X$ is termed as the forecast error. The value of σ is used to control the value of the forecast error.

Latin hypercube sampling (LHS) strategy is used to sample δ . A value of $\sigma = 0.12$ (i.e 12% forecast error) was used to generate the set of time series for training and testing.

The hyperparameters for the DDPG algorithm are specified in table 6.2. These are held constant through all runs of the simulation.

Fig.6.1 illustrates the learning process of the networks used within the DDPG algorithm. The updation occurs with the goal of maximizing the expected value of the discounted accumulated rewards as mentioned in eq.4.8. The performance of the learned policy is evaluated on the test data

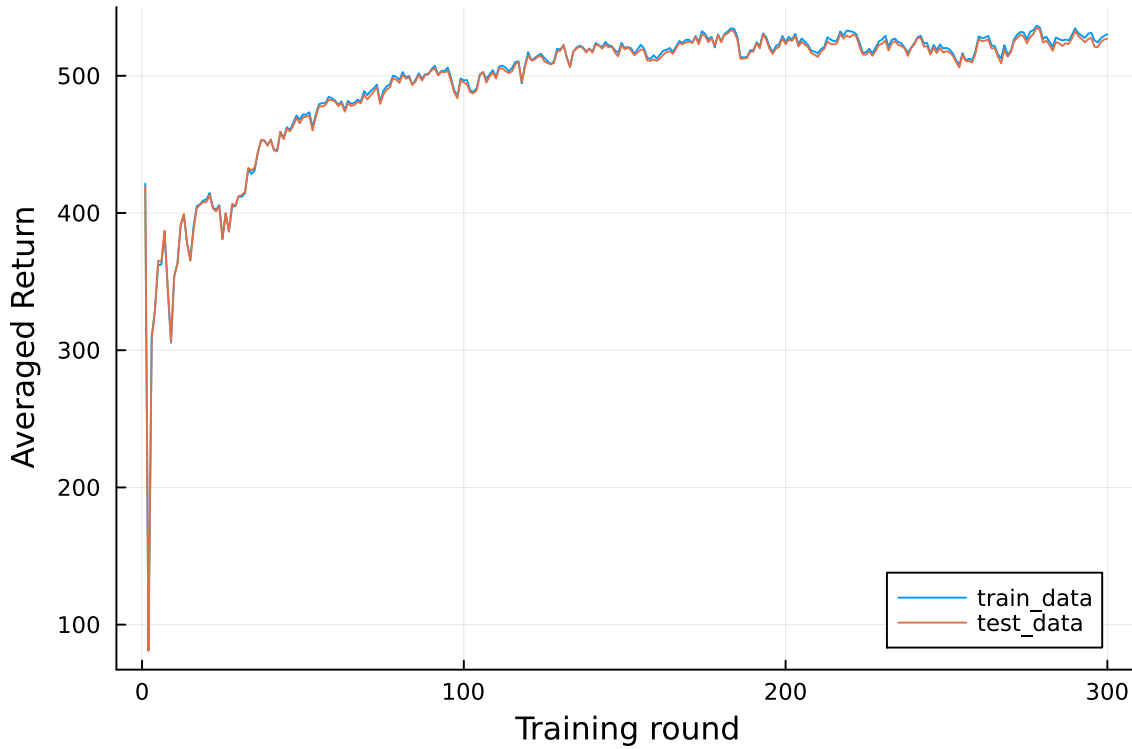
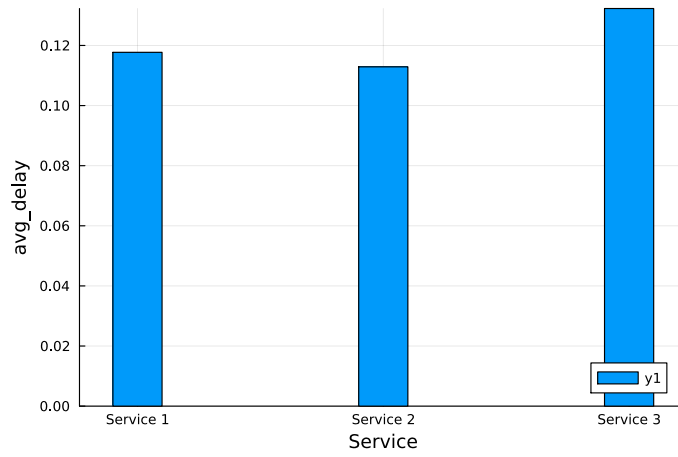


Figure 6.1: Learning

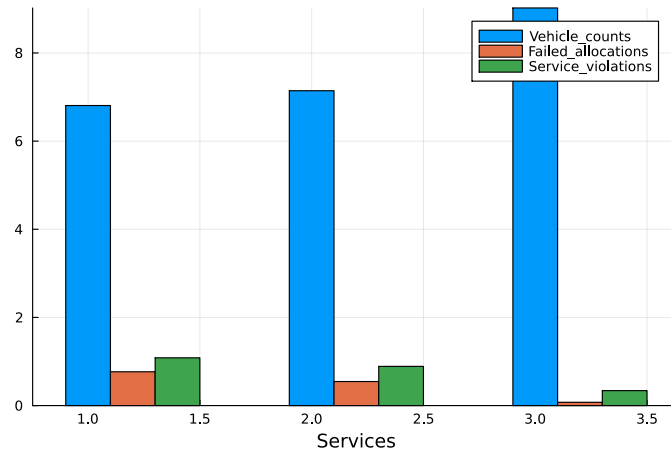
set. Fig.6.2 shows various metrics that are used to judge the quality of the decisions made by the policy.

Fig.6.2b captures the number of vehicles whose service delay exceeded the max threshold. It also mentions the number of vehicles who failed to receive any compute or bandwidth resources. By utilizing the figures in fig.(6.2 we can infer the QoS of the user and hence judge the quality of the actions.

We can also measure the trends observed with respect to resource allocations. Fig.6.3 shows the average number of BRBs and CRBs allocated to each service type.



(a) Average service delays



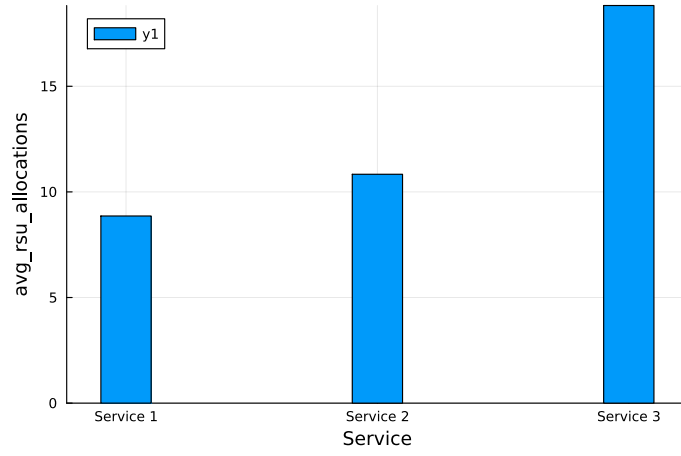
(b) QoS

Figure 6.2: Performance for $\lambda_v = 5, \lambda_a = 5, \mathcal{T} = 1.5$ minutes

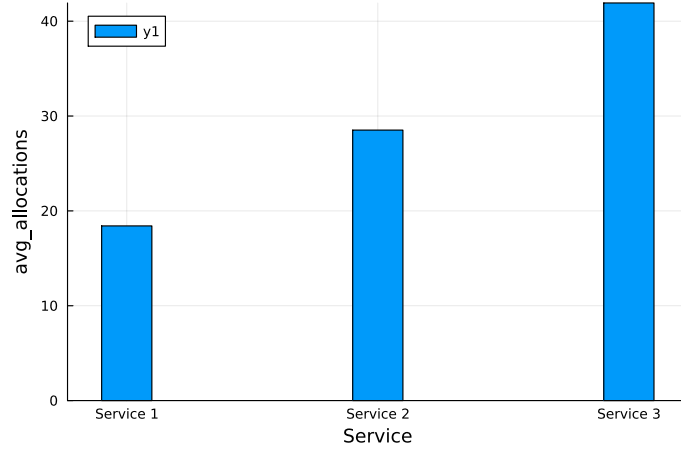
6.0.1 Biasing model behaviour : Reward Modelling

As stated in section.4 the goal of a MDP is to find a policy which aims to maximize the value of eq.(4.8). By changing the definition of the reward function, we may bias the model towards different behaviours. For instance consider the plots shown in fig.6.4 and fig.6.5. This scenario was constructed with $\lambda_v = 7, \lambda_a = 5, \mathcal{T} = 1.5$ minutes. The value of these parameters gives us an average of 43 vehicles within the simulation. We utilize the reward function defined as per eq.(4.9).

Due to the larger number of vehicles within the simulation, it is difficult to provide satisfactory service for all the vehicles. However since the reward function gives equal weight to all services, the model preferentially allocates more resources for the services with smaller compute and communica-



(a) BRB allocations



(b) CRB allocations

 Figure 6.3: Allocations for $\lambda_v = 5, \lambda_a = 5, \mathcal{T} = 1.5$ minutes

tion demands. This leads to skewing of the QoS towards the “lighter” services.

Now we consider an altered reward function given by

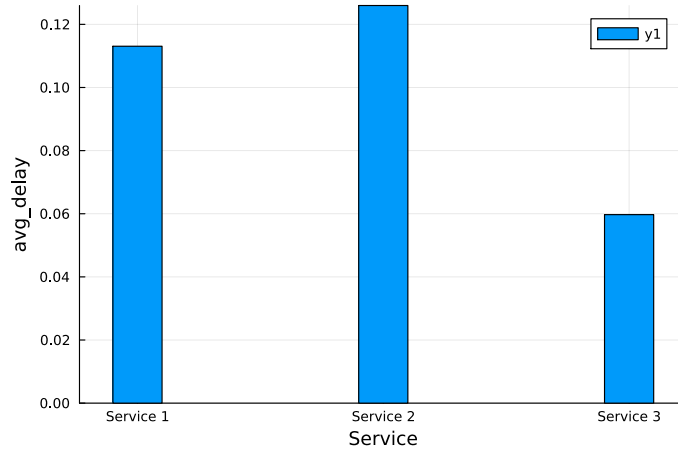
$$R_t(s_t, a_t) = \sum_{\forall j} R_{j,t}(s_t, a_t) \quad (6.2)$$

$$= \sum_{\forall j} \sum_{v \in V_j} (\mu_k U_{v,t} + \delta_v) \quad (6.3)$$

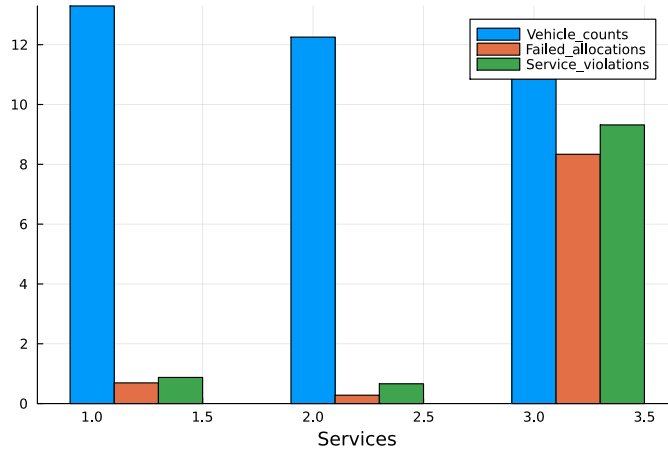
where μ_k is a multiplier for the service k and δ_v is defined as,

$$\delta_v = \begin{cases} c & \text{if } b_v > 0 \ \& \ c_v > 0 \\ -c & \text{otherwise} \end{cases} \quad (6.4)$$

where $c > 0$. The above reward function gives weighting to each service. It also gives a small positive



(a) Average service delays



(b) QoS

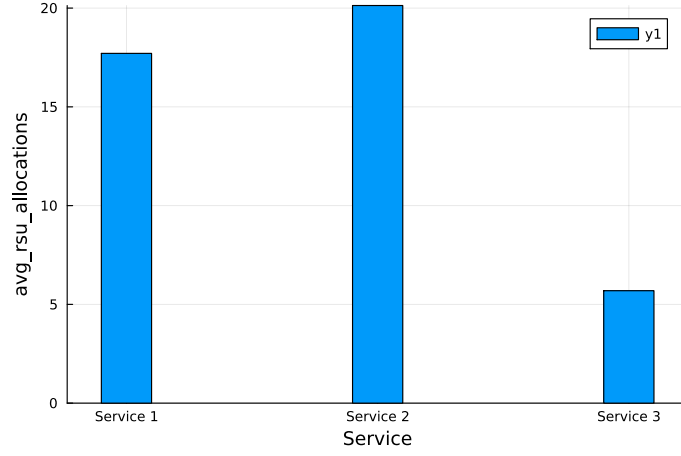
Figure 6.4: Service quality using eq.(4.2)

reward for not failing to allocate resources and a small penalty for failing to allocate resources to a vehicle. The plots in fig.6.6 gives the performance of the model using the new reward function. Fig.6.7 shows the trends in allocations. A value of $c = 0.2$ and $\mu_k = [0.5, 0.75, 1]$ was used.

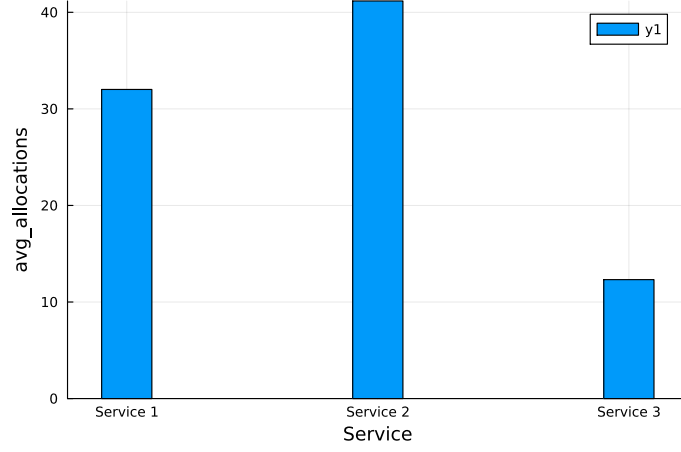
We now observe that the allocations align with the demands of the service. We also observe that the QoS is equally spread out for all services.

6.0.2 A study on Migrations

Service migration as described in section.3.2 is a method to ensure constant access to a service whilst maintaining minimal service delays occurring due to communications. Service migration can also be used to balance load across servers. In this section we will study the effect of migrations over QoS.



(a) BRB allocations



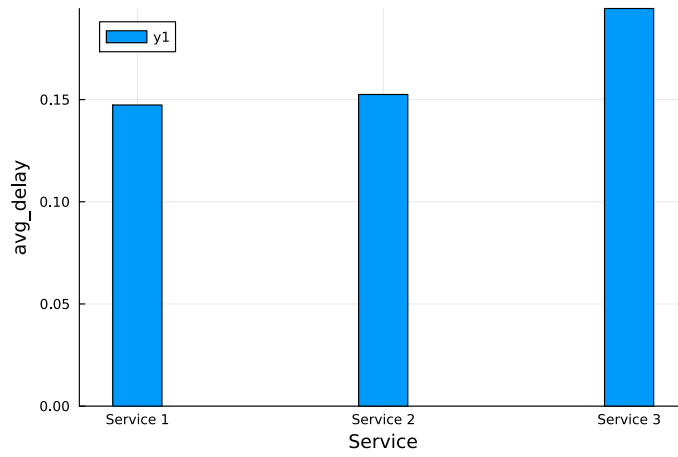
(b) CRB allocations

Figure 6.5: Allocations made by using eq.(4.2)

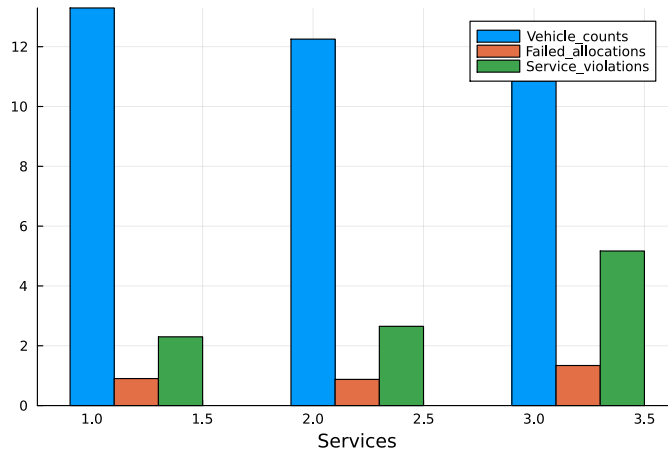
We will also look at how the number of migrations are affected by the time threshold parameter \mathcal{T} . We assume that all services offload data of the same size $L_k = 7 \text{ MB } \forall k$. We also assume equal compute requirements for each service $c_k = 0.4 \times 10^9 \text{ cycles } \forall k$. This reduces the problem to that of a homogenous service set however the analysis can still be extended to a heterogenous service set by using methods such as priority for migrations, grouping migrations based on services, reward modeling as described in section.6.0.1 etc.

Fig.6.8 shows the performance of the model for the following parameters $\lambda_v = 5, \lambda_a = 15, \mathcal{T} = 1.5$ minutes. This gives us an average of 30 vehicles within the simulation.

The QoS plots in fig.6.8 migration fraction shown in fig.6.12 clearly shows that in this case migrations are not of importance as the server is able to allocate resource to all vehicles without



(a) Average service delays



(b) QoS

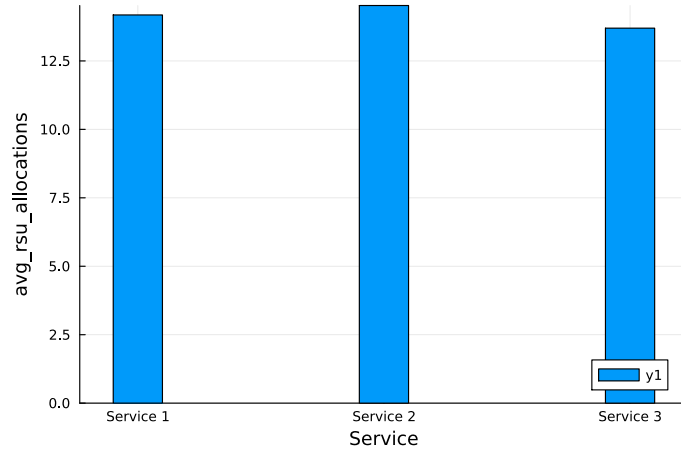
Figure 6.6: Service quality using eq.(6.3)

being overloaded.

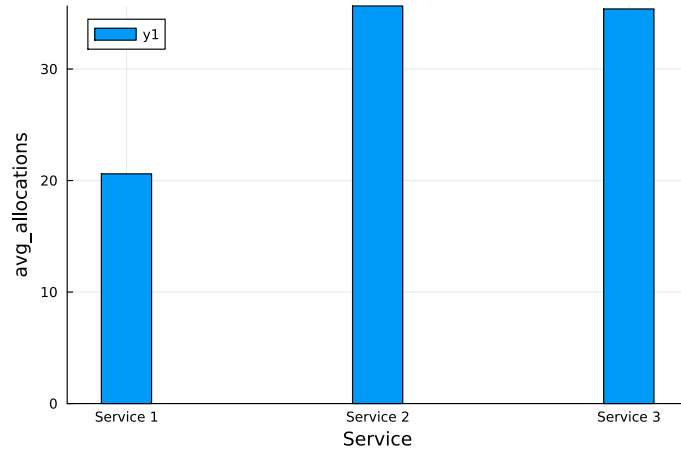
Now we consider the case of $\lambda_v = 7, \lambda_a = 15, \mathcal{T} = 1.5$ minutes. This gives us an average of 43 vehicles in the simulation.

Fig.6.9 now shows that the server is no longer able to allocate sufficient resources for the vehicles. We also see in fig.6.12 that the server is trying to maximize the allocations by migrating all available candidates for migrations. However the number of available candidates is very small. We can increase the possible candidates for migrations by increasing the value of \mathcal{T} . Fig.6.10 shows the performance for $\mathcal{T} = 3$ minutes.

Now the server is able to make greater number of migrations and hence able to improve the QoS marginally. However fig.6.12 shows that the server is migrating only a small fraction of due to the



(a) BRB allocations

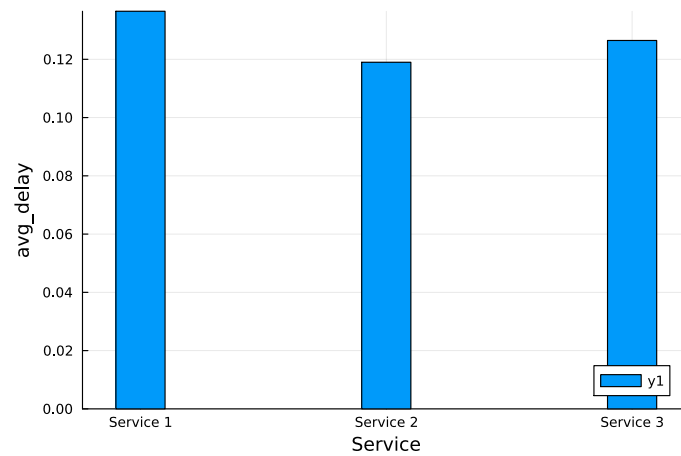


(b) CRB allocations

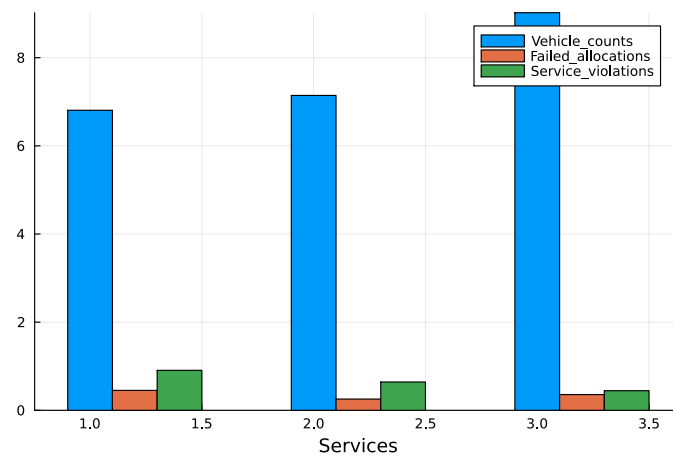
Figure 6.7: Allocations made by using eq.(6.3)

smaller number of allocations made by server $i + 1$ for the migrations. We can see the effect of increasing the allocations in fig.6.11 where we increase the value of λ_a to 20.

Fig.6.11 and fig.6.12 shows that the server is able to make greater number of allocations (hence fewer Failed allocations) by migrating a greater fraction of users.



(a) Average service delays



(b) QoS

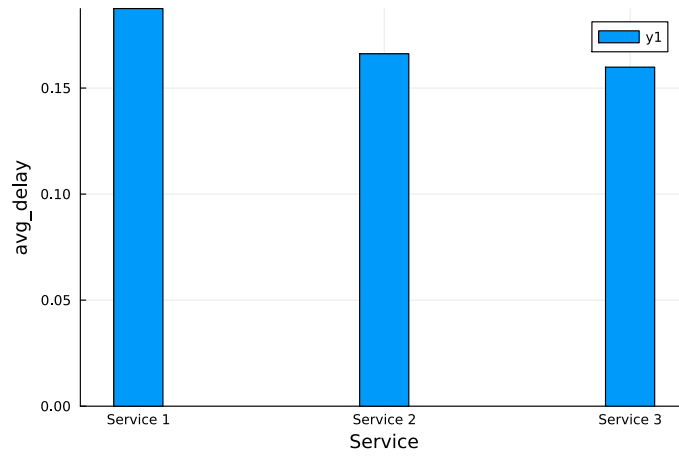
Figure 6.8: Effect of migrations on QoS

Table 6.1: Parameter Definitions and Values

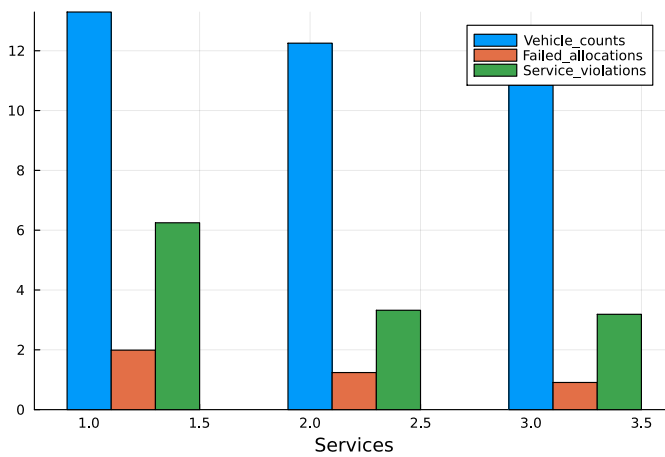
Parameter	Value
Mean vehicle speed μ_v in $\frac{\text{km}}{\text{hr}}$	32
Standard deviation of speeds σ_v	4
Lower and upper limits of speed (v_l, v_u)	(24,40)
Migration rate λ_m	5
RSU coverage length in km	1
Service set $k \in \mathcal{K}$	(1,2,3)
Service sizes in MB	(5,7,10)
Delay threshold τ in s	0.35
Maximum delay threshold τ_{\max} in s	0.5
Compute cycles required by service c_k ($\times 10^9$)	(0.2,0.3,0.4)
Size of a bandwidth resource block B in kHz	180
Total number of BRBs available at RSU j	50
Size of a CRB in cycles ($\times 10^9$)	1
Total number of CRBs available at edge server	100
Transmit power P in mW	200
Channel gain h	3
Path loss exponent α	3.75
Power control factor ϵ	0.25
Reference distance d_o in m	500
Noise spectral density in $\frac{\text{dBm}}{\text{Hz}}$	-174

Table 6.2: Hyperparameters

Parameter	Value
Optimizer : ADAM($\eta, \beta_1, \beta_2, \epsilon$)	(0.01, 0.9, 0.999, 10^{-8})
Discount parameter γ	0.98
Soft update parameter τ	0.01
Replay buffer size	2000
Minibatch size	120

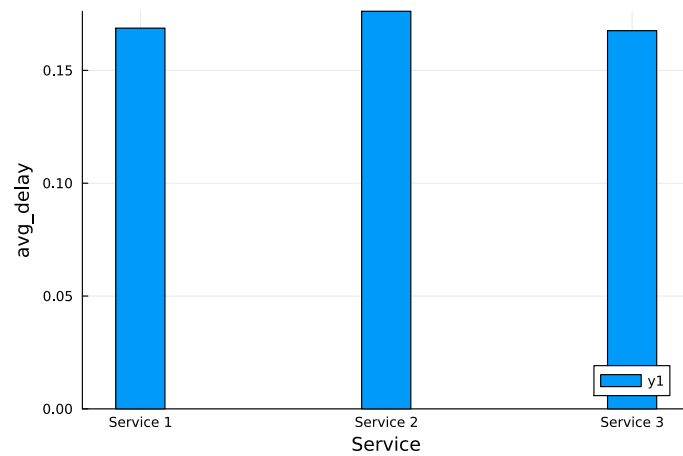


(a) Average service delays

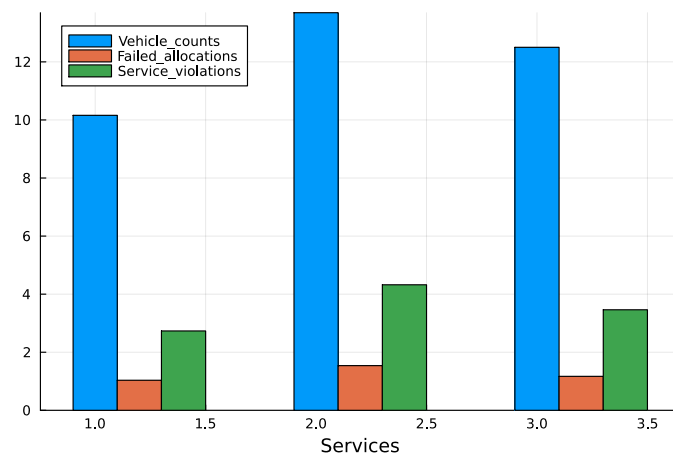


(b) QoS

Figure 6.9: Effect of migrations on QoS

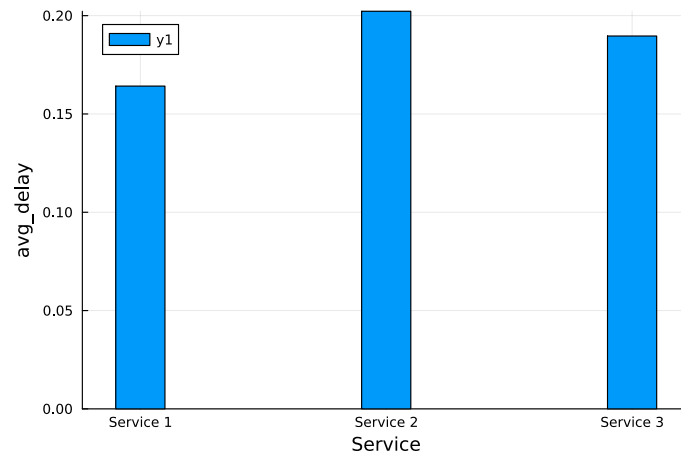


(a) Average service delays

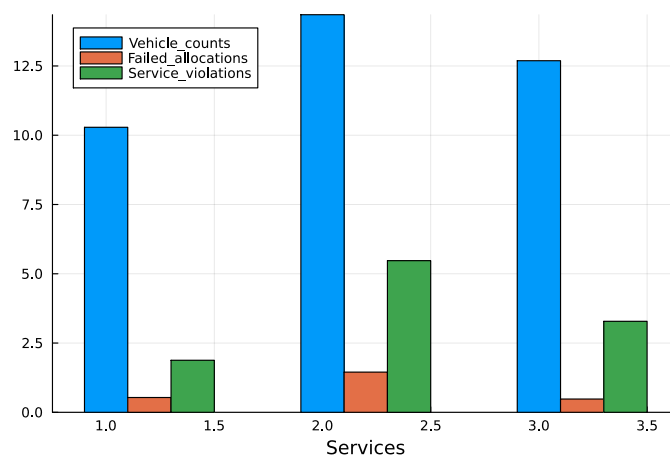


(b) QoS

Figure 6.10: Effect of migrations on QoS



(a) Average service delays



(b) QoS

Figure 6.11: Effect of migrations on QoS

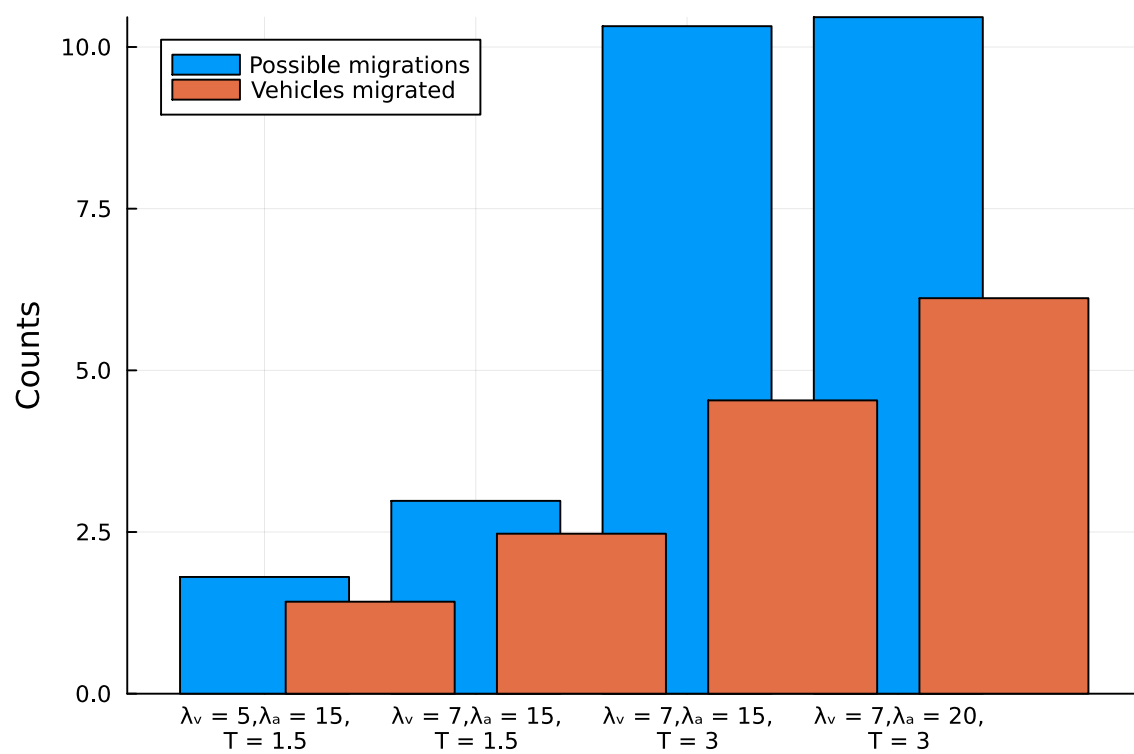


Figure 6.12: Migration candidates and migration fractions

Chapter 7

Conclusion and Future Work

In this project, we considered the computation offloading problem in dynamic mobile environment. Literature survey suggested and highlighted various challenges and methods to get optimal Quality of Service and minimum service delay. We mainly focused on Service Migration in this work. We broadly classified service migration methods into two methods, Network Initiated Migration and Vehicle Initiated Migration. Referring to the existing works, we decided to implement network initiated migration due to its global scope of the environment and user grouping ability. We also highlighted the adaptive bandwidth and compute resource allotment was missing in the existing works. We implement a dynamic environment which include vehicles, RSUs, MEC servers and services to simulate the offloading process. We formulate a Markov Decision Process (MDP) using this environment. We make a single MEC server the Agent and feed the relevant state vectors to it learn the mobility, service request pattern of the vehicles etc. We use a DDPG trained DRL model to take resource allotment and service migration decisions. Finally, we present simulation results which projects our system performance and meeting optimal service delay requirements.

We aim to scale this problem to Multi-agent scenario. As for this project, we considered a single agent (MEC sever) being fed simulated data from the neighboring MEC servers. In the future, attempts to incorporate Multi Agent Deep Reinforcement Learning (MADRL) is in process. We will also attempt to model complex road networks taken from smart cities to deploy our agents and measure the performance accordingly.

Appendix A

7.1 Components of a RL based system

Any RL setup can be reduced down to three interoperating components.

1. Agent
2. Environment
3. Reward

7.1.1 Agent

The agent is the component of the RL setup which performs the process of sequential decision making. Its goal is to maximize the total future reward.

7.1.2 Environment

The environment is the ‘playground’ through which the agent enacts its actions. The environment also provides the feedback to the agent via a reward signal as shown in ???. The agent environment interaction can be summarized as follows:

- At each step t , the agent:
 1. Executes action A_t .
 2. Receives observation O_t .
 3. Receives scalar reward R_t .
- The environment:

1. Receives action A_t .
2. Emits observation O_{t+1} .
3. Emits scalar reward R_{t+1} .

7.1.3 Reward

A reward R_t is a scalar feedback signal. It indicates how well an agent is doing at a step t . RL is based on the *reward hypothesis*:

All goals can be described by the maximization of cumulative reward.

So we would like our rewards to encode the following characteristics:

- Actions may have long term consequences.
- Rewards may be delayed.
- It may be better to sacrifice immediate reward to gain more long-term rewards.

7.2 History and State

The *history* is a sequence of observations, actions and rewards i.e all observable variables up to time t .

$$H_t = O_1, R_1, A_1 \dots, A_{t-1}, O_t, R_t$$

The history determines what will happen next in the given process, more specifically it is the *state* which determines what happens next. State is a function of history i.e:

$$S_t = f(H_t)$$

7.2.1 Environment State

The environment state S_t^e is the environment's private representation of the events to come i.e it is whatever data the environment uses to pick the next observation/reward. This state is not usually visible to the agent and even if it is, environment state may contain irrelevant information.

In the case of fully observable environments the agent is able to directly observe S_t^e i.e:

$$O_t = S_t^a = S_t^e$$

7.2.2 Agent State

The agent state S_t^a is the agent's internal representation of the environment i.e it is whatever data the agent uses to pick the next observation/reward.

7.3 Components of an RL agent

An RL agent may include one or more of the following:

- Policy
- Value function
- Model

7.3.1 Policy

A *policy* is a map from state to action i.e it mathematical descriptor of an agent's behaviour. A policy can be deterministic or stochastic. Policies are denoted with π .

7.3.2 Value function

A *value* function is a measure of the prediction of future rewards. It is used to evaluate the quality of a state and hence can be used to select between actions.

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s]$$

7.3.3 Model

A *model* is a representation of the environment created by the agent.

7.3.4 Types of RL agents

Based on the presence or absence of the above components, an RL agent can be classified into the following fields:

- Value based
 - Policy
 - Value function
- Policy based
 - Policy
 - Value function
- Actor Critic
 - Policy
 - Value function

7.4 Markov Decision Process

Markov Decision Processes (MDP) are a formal definition of fully observable environments in the setting of RL. To build an understanding of MDPs we must first begin by learning the building blocks of MDPs which are MP and MRPs.

7.4.1 Markov Process

A Markov process is a memoryless random process, i.e a sequence of random states S_1, S_2, \dots with the *Markov property*.

Markov Property:

A state S_t is Markov iff

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

“The future is independent of the past given the present.”

The state captures all relevant information from the history and once the state is known the history can be thrown away.

State Transition Matrix:

For a Markov state s and successor state s' , the state transition matrix is defined as:

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

So this matrix defines the transition probabilities from all states s to all successor states s' .

Given the above definition of Markov property it is clear that a Markov process can be cleanly represented by the state space S and the probability transition matrix P .

A Markov process is a tuple $\langle S, P \rangle$.

7.4.2 Markov Reward Process

The Markov chain has dealt only with a Stochastic environment. Let us now bring it closer to the problem of RL by including one of the components : **rewards**.

A MRP is a Markov chain with *values*.

A Markov Reward Process is a tuple $\langle S, P, R, \gamma \rangle$.

Where R is the reward function and $\gamma \in [0, 1]$ is called the discount factor.

Note the changes between ?? where now each arc contains a reward upon exiting the state.

Return:

The return G_t is the total discounted reward from time step t .

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

. The discount factor γ is a measure of the “value” of receiving reward R after $k + 1$ time steps from t .

γ close to 0 leads to “myopic” evaluation (value immediate rewards, do not care about future rewards.)

γ close to 1 leads to “far-sighted” evaluation (weight rewards equally and consider the scope of all rewards from this time step onwards).

Value Function:

The value function $v(s)$ gives the long-term value of a state s . The state value function of an MRP is the expected return starting from state s .

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

Bellman Equation for MRPs

The value function can be decomposed into two parts.

- immediate reward R_{t+1} .
- discounted value of successor state $\gamma v(S_{t+1})$.

$$v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$$

The Bellman equation can be expressed more concisely using matrices:

$$v = R + \gamma P v$$

Which gives us a simple linear equation that can be solved as:

$$v = (I - \gamma P)^{-1} R$$

This may sound great, however the computational complexity of solving this is $O(n^3)$ given n states. So the direct solution obtained this way is only feasible for small MRPs. The use of iterative methods is required for large MRPs

7.4.3 Markov Decision Process

The MRP came closer to the discription of an RL setting, however it is missing one key element that will enable it to describe an RL setting i.e agency or *actions*. An MDP is an MRP with decisions/actions.

A Markov Decision Process is a tuple $\langle S, A, P, R, \gamma \rangle$.

Where A is a finite set of actions. P is the state transition matrix. Now every action must have its own state transition matrix.

$$P_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

R is the reward function which may also depend on the action taken from a state s .

$$R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Policies:

A policy π is a distribution over actions given states,

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

This is a stochastic policy.

A policy fully defines the behaviour of an agent.

MDP policies depend only on current state (they follow the Markov property) i.e the policies are stationary (time-independent).

Reduction of MDP to an MP or MRP

Given an MDP $M = \langle S, A, P, R, \gamma \rangle$ and a policy π , we can reduce it down to:

- an MP $\langle S, P^\pi \rangle$
- an MRP $\langle S, P^\pi, R^\pi, \gamma \rangle$

where P^π and R^π are the averaged transition probability and rewards over the policy π .

Value functions:

- The state value function $v_\pi(s)$ of an MDP is the expected return starting from state s whilst following the policy π .
- The action value function $q_\pi(s, a)$ is the expected return starting from state s , taking an action a , and then following policy π .

Bellman Equations:

Once again, the Bellman equation can be used to decompose both the state and action value functions into the immediate reward and the value of the state you will end up in.

Bellman equation for v_π

Bellman equation for q_π

Bellman equation for q_π

The above options enable us to solve for the value functions in two ways:

- Use 7.4.3 to convert the MDP to an MRP and solve for v using the Bellman equations in matrix form.
- Use dynamic programming to do a look ahead of the value functions.

the second solution is preferred because of the $O(n^3)$ time complexity of the first solution.

Optimal Value Function:

The optimal state value function $v_*(s)$ is the maximum value function over all policies.

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

The optimal action value function $q_*(s, a)$ is the maximum action-value function over all policies.

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

The optimal value function specifies the best possible performance in the MDP. The MDP is solved when the optimal action value function is known because now you choose the actions which with the highest value of q_* .

List of Figures

2.1	Left: Vehicle Initiated Migration, Right: Network Initiated Migration	7
3.1	Comparison of our work with [6] and [5]	10
3.2	System Model Overview	11
3.3	Workflow of the system	13
3.4	Service Migration	14
3.5	System Integrated with DRL Model	15
5.1	DDPG Algorithm	22
6.1	Learning	25
6.2	Performance for $\lambda_v = 5, \lambda_a = 5, \mathcal{T} = 1.5$ minutes	26
6.3	Allocations for $\lambda_v = 5, \lambda_a = 5, \mathcal{T} = 1.5$ minutes	27
6.4	Service quality using eq.(4.2)	28
6.5	Allocations made by using eq.(4.2)	29
6.6	Service quality using eq.(6.3)	30
6.7	Allocations made by using eq.(6.3)	31
6.8	Effect of migrations on QoS	32
6.9	Effect of migrations on QoS	34
6.10	Effect of migrations on QoS	35
6.11	Effect of migrations on QoS	36
6.12	Migration candidates and migration fractions	37

List of Tables

3.1	Notation and Brief Description Table	16
5.1	Network Parameters (Actor)	21
5.2	Network Parameters (Critic)	23
6.1	Parameter Definitions and Values	33
6.2	Hyperparameters	34

References

- [1] Y. Kim, N. An, J. Park, and H. Lim, “Mobility support for vehicular cloud radio-access-networks with edge computing,” in *2018 IEEE 7th International Conference on Cloud Networking (Cloud-Net)*, Oct 2018, pp. 1–4.
- [2] W. Bao, C. Wu, S. Guleng, J. Zhang, K.-L. A. Yau, and Y. Ji, “Edge computing-based joint client selection and networking scheme for federated learning in vehicular iot,” *China Communications*, vol. 18, no. 6, pp. 39–52, 2021.
- [3] M. S. Bute, P. Fan, G. Liu, F. Abbas, and Z. Ding, “A collaborative task offloading scheme in vehicular edge computing,” in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–5.
- [4] S. Khamari, A. Rachedi, T. Ahmed, and M. Mosbah, “Adaptive deep reinforcement learning approach for service migration in mec-enabled vehicular networks,” in *2023 IEEE Symposium on Computers and Communications (ISCC)*, 2023, pp. 1075–1079.
- [5] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, “A joint service migration and mobility optimization approach for vehicular edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9041–9052, 2020.
- [6] W. Chen, M. Liu, F. Wu, H. Wu, Y. Miao, F. Lyu, and X. Shen, “Msm: Mobility-aware service migration for seamless provision: A data-driven approach,” *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 15 690–15 704, 2023.

- [7] H. Peng and X. Shen, “Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks,” *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2416–2428, 2020.
- [8] Y. Liu, H. Yu, S. Xie, and Y. Zhang, “Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks,” *IEEE Transactions on Vehicular Technology*, vol. PP, pp. 1–1, 08 2019.
- [9] R. Sharma, K. Gummaraju, P. Anantharam, O. Saraf, and V. K. Tumuluru, “Decentralized computation offloading in mobile edge computing systems,” in *2021 International Conference on Smart Applications, Communications and Networking (SmartNets)*, 2021, pp. 1–6.
- [10] Y. Liu, H. Yu, S. Xie, and Y. Zhang, “Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks,” *IEEE Transactions on Vehicular Technology*, vol. PP, pp. 1–1, 08 2019.
- [11] Y. Miao, F. Lyu, F. Wu, H. Wu, J. Ren, Y. Zhang, and X. S. Shen, “Mobility-aware service migration for seamless provision: A reinforcement learning approach,” in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 5064–5069.
- [12] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, S. Pang, H. Liu, Y. Qin, and P. Chen, “Mobility-aware and migration-enabled online edge user allocation in mobile edge computing,” in *2019 IEEE International Conference on Web Services (ICWS)*, 2019, pp. 91–98.

The demand for computing resources

ORIGINALITY REPORT

15%

SIMILARITY INDEX

12%

INTERNET SOURCES

8%

PUBLICATIONS

8%

STUDENT PAPERS

PRIMARY SOURCES

1	www.slideshare.net Internet Source	1%
2	ieeexplore.ieee.org Internet Source	1%
3	cse.iitkgp.ac.in Internet Source	1%
4	zackkhan.github.io Internet Source	1%
5	2ez4ai.github.io Internet Source	1%
6	Submitted to Hong Kong University of Science and Technology Student Paper	1%
7	Lusungu Josh Mwasinga, Duc-Tai Le, Syed M. Raza, Rajesh Challa, Moonseong Kim, Hyunseung Choo. "RASM: Resource-Aware Service Migration in Edge Computing based on Deep Reinforcement Learning", Journal of Parallel and Distributed Computing, 2023 Publication	1%