COMPUTATION OFFLOADING IN MOBILE EDGE COMPUTING: AN OPTIMAL STOPPING THEORY APPROACH

IBRAHIM ALGHAMDI

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

SCHOOL OF COMPUTING SCIENCE

COLLEGE OF SCIENCE AND ENGINEERING
UNIVERSITY OF GLASGOW

OCTOBER 2021

Abstract

In recent years, new mobile devices and applications with different functionalities and uses, such as drones, Autonomous Vehicles (AV) and highly advanced smartphones have emerged. Such devices are now able to launch applications such as augmented and virtual reality, intensive contextual data processing, intelligent vehicle control, traffic management, data mining and interactive applications. Although these mobile nodes have the computing and communication capabilities to run such applications, they remain unable to efficiently handle them mainly due to the significant processing required over relatively short timescales. Additionally, they consume a considerable amount of battery power. Such limitations have motivated the idea of computation offloading where computing tasks are sent to the Cloud instead of executing it locally at the mobile node. The technical concept of this idea is referred to as Mobile Cloud Computing (MCC). However, using the Cloud for computational task offloading of mobile applications introduces a significant latency and adds additional load to the radio and backhaul of the mobile networks. To cope with these challenges, the Cloud's resources are being deployed near to the users at the Edge of the network in places such as mobile networks at the Base Station (BS), or indoor locations such as Wi-Fi and 3G/4G access points. This architecture is referred to as Mobile Edge Computing or Multi-access Edge Computing (MEC). Computation offloading in such a setting faces the challenge of deciding which time and server to offload computational tasks to.

This dissertation aims at designing time-optimised task offloading decision-making algorithms in MEC environments. This will be done to find the optimal time for task offloading. The random variables that can influence the expected processing time at the MEC server are investigated using various probability distributions and representations. In the context being assessed, while the mobile node is sequentially roaming (connecting) through a set of MEC servers, it has to locally and autonomously decide which server should be used for offloading in order to perform the computing task. To deal with this sequential problem, the considered offloading decision-making is modelled as an optimal stopping time problem adopting the principles of Optimal Stopping Theory (OST).

Three assessment approaches including simulation approach, real data sets and an actual implementation in real devices, are used to evaluate the performance of the models. The results indicate that OST-based offloading strategies can play an important role in optimising the task offloading decision. In particular, in the simulation approach, the average processing time achieved by the proposed models are higher than the Optimal by only 10%. In the real data set, the models are still near optimal with only 25% difference compared to the Optimal while in the real implementation, the models, most of the time, select the Optimal node for processing the task. Furthermore, the presented algorithms are lightweight, local and can hence be implemented on mobile nodes (for instance, vehicles or smart phones).

Acknowledgements

AlhamduliLlahi Robil 'Alamin!

First and foremost, all praise is due to God, who advised me to start this journey and assisted me throughout.

I am grateful to my supervisors, Prof. Dimitrios P. Pezaros and Dr. Christos Anagnostopoulos, for the tremendous support they have provided to me and their guidance and patience over the years.

I wish to acknowledge Al-Baha University for providing me with a scholarship, as well as the additional funding provided for my travel during my studies. Furthermore, I am thankful for the support I received from the Saudi Arabian Cultural Bureau in the UK.

I extend my thanks to my friends, including Dr.Stephen McQuistin, Dr.Blair Archibald and Haruna Umar Adoga, with whom I have shared an office. We had very productive discussions while I was studying for my PhD, while they also made my PhD more enjoyable. I wish to extend my special thanks to my cousin, Khaled Alghamdi for his support and motivation over the years. Additionally, I thank my friends from Saudi Arabia who were present in Glasgow, including: Adel Alghamdi, Abdullah Alghamdi, Saad Alahmari and Abdulwhab Alkharashi for alleviating the stress of my PhD study.

I am grateful to my PhD examiners Dr. Jose Cano Reyes (University of Glasgow) and Dr. Nikos Tziritas (University of Thessaly) for providing feedback on this thesis.

Finally, and most importantly, I thank my family and specifically my father and mother for their encouragement and unlimited support. Enormous gratitude goes to my wife for her love, unconditional understanding and support that she provided during my PhD journey. I extend a special thank to my siblings for their continuous encouragement and support.

وَقُلْ رَبِّ زِدْنِي عِلْمًا ”But say, O my Lord! advance me in knowledge”,
Quran, Taha, Verse [135]

Table of Contents

List of Tables

List of Figures

Acronyms

1D One Dimensional mobility model

AHP Analytical Hierarchy Process

AV Autonomous Vehicle

BCP Best-Choice Problem based offloading

BS Base Station

COT Cost-Based Task Offloading

CPU Central Processing Unit

DNN Deep Neural Network

DQL Deep Q Learning

DRL Deep Reinforcement Learning

DSRC Dedicated Short-Range Communication

DTC Delay Tolerant Computing

DTO Delay-tolerant Task Offloading

ETSI European Telecommunications Standards Institute

GPS Global Positioning System (GPS)

HD High Definition

HMM Hidden Markov Model

IP Internet Protocol

ISG Industry Specification Group

KDE Kernel Density Estimation

LAN Local Area Network

LSTM Long Short Term Memory

MAB Multi-Armed Bandit

MAPE-k Monitoring, Analysing, Planning and Executing

MCC Mobile Cloud Computing

MCS Mobile Crowd Sensing

MDP Markov Decision Processes

MEC Mobile Edge Computing

OST Optimal Stopping Theory

P2P Peer to Peer network

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RL-QL Reinforcement Learning-based Q-Learning

RL-SARSA Reinforcement-Learning-based State-Action-Reward-State-Action

RSU Road Side Units (RSUs)

RTT Round Trip Time (RTT)

SCC Small Cell Cloud

SDN Software Defined Network

UAV Unmanned Aerial Vehicle

V2I Vehicle to Infrastructure

V2V Vehicle to Vehicle communication (V2V)

VeFN Vehicular Fog Network (VeFN)

VM Virtual Machine

Chapter 1

Introduction

1.1 Overview

Mobile-Edge Computing (MEC) is a computing paradigm that optimises the Cloud's resources by bringing applications and content closer to users at the Edge of the network within the Radio Access Network (RAN) [1]. MEC involves deploying small servers or data centres in places like Road Side Units (RSUs), Base Stations (BSs) and access points. MEC servers can be an advantage for various types of mobile nodes and their applications including computation offloading for intensive applications such as augmented reality [2, 3]. Further, MEC servers can be an intermediate data-processing layer for data offloaded by mobile nodes [4].

As proposed in [5, 6], MEC servers can be deployed within RSUs, and it is expected to provide computing resources for tasks offloaded by mobile nodes as mobile nodes pass by. The mobile node may be either a passenger on board who is running different applications or it can be the smart vehicle itself. As an example, MEC servers within the RSUs can play an important role in enabling performance improvements for mobile vehicular terminals such as Autonomous Vehicles (AV) [7]. The AV can run intelligent vehicle control, traffic management and interactive applications using the built-in computation units. AVs are equipped with a massive number of sensors that collect contextual data for different types of applications such as transportation systems and navigation applications [5]. The AVs can collect and sense contextual data and apply different algorithms for data analytics tasks. An autonomous driving vehicle, for example, produces and consumes approximately 40 terabytes of data per eight driving hours (e.g., a city's High Definition (HD) map is approximately 1.5TB) [8]. However, despite AVs typically include on-board units, they have small-scale computing and storage capabilities and hence are dependent on other computational servers [9]. Also, such applications may require significant computation resources and constrained time delays [5]. Similarly, mobile devices such as smartphones, tablets, and netbooks run advanced applications, such as gaming, virtual reality, and natural language processing, which

do not match the capabilities and battery power of these mobile devices [10]. Therefore, the mobile node then would be required to offload such tasks to one of the MEC servers to enhance its resource capabilities and to meet the applications' requirements.

When offloading, the key problem is the offloading decision by which the mobile node selects an Edge server to offload the computing task to as the MEC servers' load have large variation. In particular, in MEC environments, it is expected that sometimes there is a large number of users concurrently using the same server, whereas some other times only a few users are connecting [11, 12]. In other words, the workload of such servers may be different over time [6, 11, 12]. The selection of *where* (which MEC server) and *when* (time) to offload has a significant impact on satisfying the requirements of the offloaded tasks [9, 13, 14]. Meanwhile, computing tasks or data gathered by mobile nodes (for instance AVs) tend to have strict timeliness requirements which may result in the data becoming out-of-date [15]. Further, the existing studies in the area of computation offloading have mainly focused on the decision of whether the computing task should be offloaded or executed locally and a few studies have considered the selection of where a task should be offloaded as in [6, 16]. It is, therefore, vital to define rules by which the mobile node can select a suitable MEC server to be utilised for task offloading. When the task cannot be executed locally and as the mobile node moves, should the mobile node offload immediately or would rather delay the offloading in order to find a superior MEC server in terms of computing performance?

As an example, in the AV or in ultra-dense network use cases and in a naive centralised offloading method, the mobile node would request information from a centralised server about all the available MEC servers that could be used to offload [6, 17, 18, 19]. The centralised server would be connected to the MEC servers by wired network connectivity and can be located in the Cloud [6, 18]. However, such an architecture is not visible because it may introduce load on the network as the number of mobile nodes increases [6]. In AV use case, another solution would be using Vehicle to Vehicle communication (V2V) as discussed in [5], where a vehicle can send tasks to the MEC server (routed via vehicles) that it will pass by at the time the computational task finishes. However, this solution is dependent on other vehicles being present en route at all times, which is not guaranteed. If the mobile node only knows about the MEC server it is observing and previous methods are not available, the challenge is therefore to identify how to optimise the choice of MEC server, particularly if the mobile node has incomplete information and lacks global information about all the potential MEC servers that could be used for task offloading.

Two pivotal factors should be considered that could delay offloading until a better MEC server is found. First, mobility and speed could be considered beneficial in terms of optimising the decision of which MEC server to offload to as there is a greater likelihood of finding a better server. In other words, as the speed of the vehicle increases, the probability of having better MEC server with low workload increases [20]. Second, since the computational

task usually has a given deadline, the decision maker can use the time before the deadline to investigate offloading options and delay the offload until a better MEC server is found [6, 21].

In this thesis, the problem of deciding which server to offload to (or a time to offload at) is modelled as an Optimal Stopping Theory (OST) problem. An OST problem involves deciding when to carry out an action on the basis of a random variable that is observed in sequence for the purpose of increasing the expected payoff or reducing the expected cost [22]. Best Choice problem, the Odds sum algorithm, the House Selling problem or the Fair Coin Problem are some of well-known OST problems [22]. A variety of OST-based models is adopted in the task's offloading decision in order to make the offloading decision in an efficient way and to optimally select when to offload in an independent manner.

Two essential objectives are studied: maximising the probability of choosing the best server or time to offload, and minimising the processing time. While both objectives seem to have similar outcomes, as we shall see in later chapters, they are different in terms of performance and independence. This thesis advocates the idea that a mobile node can make an independent offloading decision in a standalone manner without relying on another mobile node or a centralised server. Moreover, this research builds on pre-existing models by designing customised algorithms that consider the requirements of the task to be offloaded. The proposed models can, for example, take the task deadline into account, or look at the quality of the data to be offloaded in a data-oriented task. Furthermore, this work provides a detailed evaluation of these methods by using different approaches, including numerical simulation, real data sets and an implementation of the models in real devices. In general, the proposed models show near-optimal results compared to other offloading methods. Such an evaluation allows the behaviour of the models to be understood in different settings. It also helps to show how such models can be applied to real world scenarios.

1.2 Thesis Statement

Offloading decisions in MEC have recently attracted attention, since they affect Quality of Service (QoS) and resource consumption for new and emerging mobile applications. A key problem when offloading computing tasks is deciding which server to select and when the offloading should begin giving the load variance for Edge servers over time. This research asserts that, by exploiting the mobility of mobile nodes in MEC environments and the deadline of the task, the decision of where and when to offload can be optimised and can be made independently as a standalone decision-making model. To optimise such a decision, this work presents a lightweight framework using the concept of OST to be deployed in the mobile node in order to have lower processing time compared to the immediate offloading and the Random offloading methods.

1.3 Contributions

The contributions of this thesis are:

- The design of a time-optimised model to maximise the probability of offloading to the optimal server.
- The design of a time-optimised model to minimise the expected processing time when offloading. In this model, the number of observations is used as an input, and the model generates a threshold for each observation (time) which is compared with the observed random variable.
- The enhancement of the Odds OST algorithm with a timeliness function; the optimal decision probabilities (Odds) depend on the freshness of data collected and the current expected processing time of the Edge servers.
- The design of a cost-based time-optimised model to minimise the expected processing time when offloading to an Edge server. Rather than having the number of observations as an input, this model considers the cost of observations and generates a threshold for a given cost.
- A comparative assessment and extensive sensitivity analysis of the proposed models with other offloading methods using numerical simulation, real data sets and an implementation of the models on real devices.

1.4 Publications

The work described in this thesis has been published in the following papers:

- Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. "Optimized Contextual Data Offloading in Mobile Edge Computing". In: IFIP/IEEE International Symposium on Integrated Network Management (IM 2021), Bordeaux, France, 17-21 May 2021, (Accepted for Publication).
- Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. "Data quality-aware task offloading in Mobile Edge Computing: An Optimal Stopping Theory approach." Future Generation Computer Systems (2020).
- Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. "On the Optimality of Task Offloading in Mobile Edge Computing Environments," 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 2019, pp. 1-6.
- Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. Delay-Tolerant Sequential Decision Making for Task Offloading in Mobile Edge Computing Environments. MDPI Information 2019, 10, 312.
- Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. "Time-Optimized Task Offloading Decision Making in Mobile Edge Computing," 2019 Wireless Days (WD), Manchester, United Kingdom, 2019, pp. 1-8 - **Recipient of the Best Paper Runner Up Paper.**

1.5 Organisation of the Thesis

The remainder of this thesis is structured as follows:

- **Chapter 2** presents a technical background of the studied area and a review on the related literature. It starts by describing the traditional architecture of the Cloud and moves to emerging architectures at the Edge of the network. The Chapter presents some examples of Edge applications and use cases, and discusses the concept of computation offloading by giving a background and a review of previous studies related to it.
- **Chapter 3** presents two OST-based models for sequential decision-making computation offloading focusing on the objective of maximising the probability of offloading

to the best server. The second model can be considered as an extension of the first one considering a data quality indicator that represents how stale the data is. The Chapter presents, for each model, a brief mathematical background, derives the optimal offloading rules to be used by the mobile node, and a numerical analysis for a number of cases in the context of the considered random variable. It concludes by providing a summary of the models and how they can be used in real world scenarios.

- **Chapter 4** extends Chapter 3 by focusing on the objective of minimising the processing time when executing a task in a MEC server. The Chapter presents two models, the first one is modelled as a finite horizon OST-based problem. The second one overcomes the limitations of the first model by relaxing the assumption that the number of observations has to be known in advanced. Similar to Chapter 3, the Chapter presents the brief mathematical background for each model, derives the optimal offloading rules to be used by the mobile node, and a numerical analysis of the random variable for different probability distributions.

- **Chapter 5** presents a comprehensive evaluation of the OST-based offloading frameworks. First, through numerical simulation, the frameworks and solution methods are evaluated to determine the optimality of solutions compared to other offloading methods considering different probability distributions. Then, it extends the evaluation by using real data sets including mobility trace and real servers' utilisation. Furthermore, the Chapter presents the results when implementing and deploying the models in a real environment. After a comprehensive evaluation for the proposed models, the Chapter discuses important aspects of the proposed models when applied in the real world.

- **Chapter 6** gives a summary of the contributions and findings of this work. It revisits the thesis statement and explores potential research directions and future work. Finally, it summarises and highlights important concluding remarks. The main chapters of the thesis and their main sections are depicted in Figure 1.1.

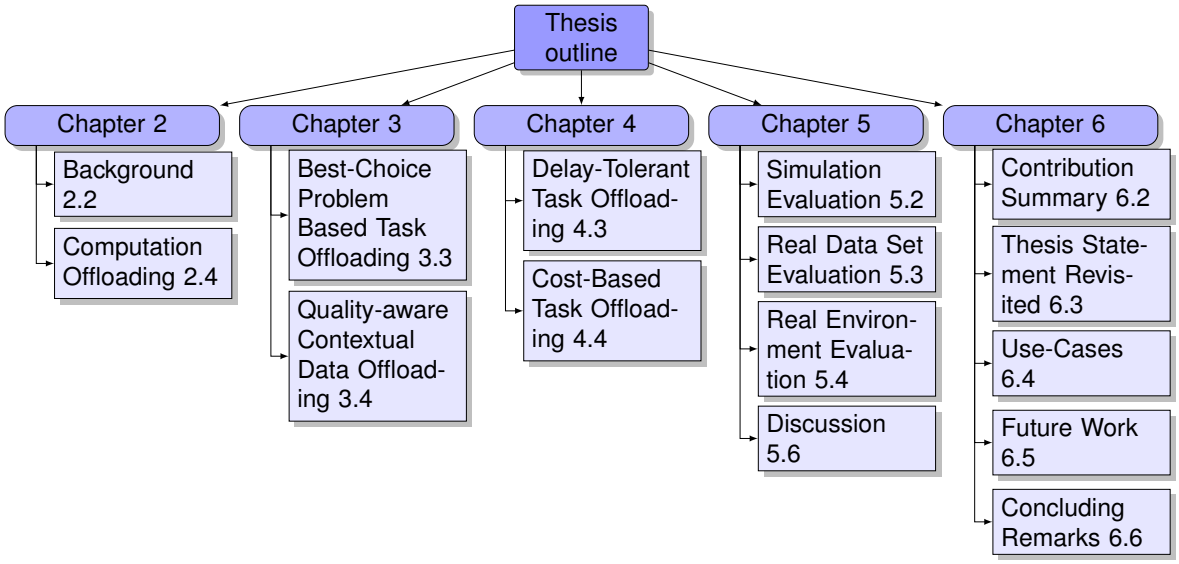


Figure 1.1: Thesis Outline.

Chapter 2

Background and Related Work

2.1 Overview

Over the last decade, there have been huge developments in mobile computing with new forms of mobile nodes being used as a computing platform for different types of applications. In addition to traditional smartphones, examples of these forms are wearable devices, drones, and smart and self-driving cars. Such developments are accelerated by advancements in the wireless communication system with high Internet speed such as 4G and 5G. However, with the appearance of a new spectrum of applications, such as augmented and virtual reality, gaming and data analysis for big data (which have high computing resource demands), these mobile nodes still encounter challenges in terms of processing and power capabilities.

The idea of computation offloading was realised from early generations of mobile devices as far back as 2002 with the concept of Cyber Foraging [23]. The main goal of Cyber Foraging is to preserve mobile devices' processing power and battery life by executing their tasks in nearby servers. Due to the introduction of the Cloud computing paradigm and its infinite resources, the development of iPhone and Android smartphones, and the proposal of the Cloudlet [24], the concept of computation offloading has been improved upon and extensively researched. The Cloud was mainly used as the destination for tasks offloaded by mobile devices. Figure 2.1 shows key breakthroughs towards the development of computation offloading.

However, as suggested by early work in computation offloading [24, 25], in order to make computation offloading beneficial, it is advised to offload tasks to nearby servers instead of using distant data centers (for example the public Cloud). Several studies have proposed new designs to deliver Cloud services to mobile nodes with different names such as Edge computing, Cloudlet, Fog computing and Mobile Edge Computing or Multi Access Edge Computing (MEC). Contrasting other proposals for Edge architecture, MEC servers are to

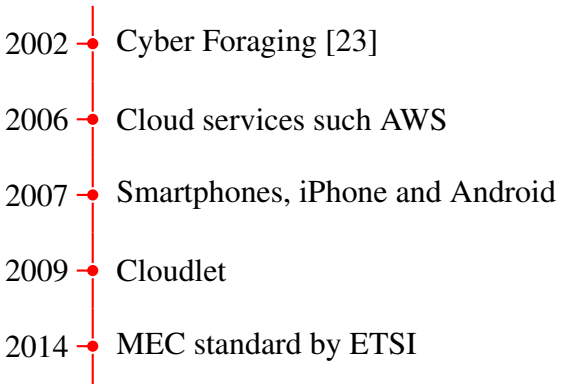


Figure 2.1: Key breakthroughs towards the computation offloading

be integrated with the current mobile network. Moreover, as the environment of MEC is dynamic and complex (in terms of mobility and the finite resources of Edge servers), resource management and decision-making over such an environment become more difficult. Therefore, different methods of optimisation have been employed to deal with the challenges involved in such an environment [26].

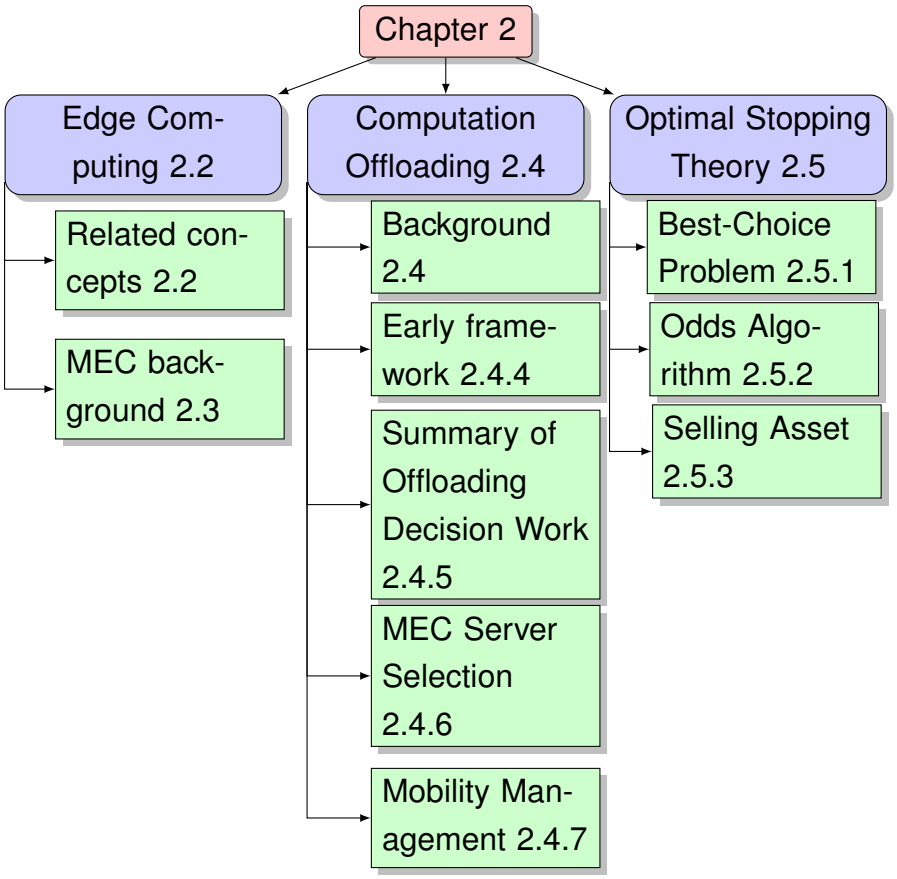


Figure 2.2: Chapter 2 outline.

In this Chapter, an outline of the related work and background of the computation offloading decision in MEC environment are presented. As this work combines MEC architecture with

computation offloading, the Chapter starts with a background on the development of MEC. It then discusses the concept of computation offloading by giving a background and review of previous studies related to it. Figure 2.2 visualises the main sections of the Chapter.

2.2 Cloud Computing to Edge Paradigm

Over the last few years, Cloud computing and its associated services have developed rapidly. These developments are widespread and comprehensive, including architectural design and infrastructure location in addition to the provision of services and resources. The Cloud is a parallel and distributed system that has a number of inter-connected and virtualised computers. These resources are provisioned and presented as one or more computing resources based on service-level agreements created through agreements between the service provider and consumers [27]. Cloud computing was proposed initially to provide reliable, customised and QoS guaranteed computing and dynamic resources for end-users of dedicated computer and storage resources located in large and distant data centres [28]. More specifically, the Cloud allow users to exploit infrastructure (e.g. servers, networks, and storage), platforms (e.g. middleware services and operating systems), and software (e.g. application programs) provided by a Cloud service providers (e.g. Google and Amazon) at an economical cost [29]. The concepts of virtualisation and the Software Defined Network (SDN) and their developments have resulted in advancements in the area of Cloud computing. The original Cloud architecture relied on service providers setting up large data centres in fixed locations. These fixed locations may be in a different country to the point of origin of the end user's data and applications [28]. Cloud advantages are not limited to business applications; the applications of Cloud extend to serve the resource constrained devices such as mobile nodes and the Internet of Things (IoT) and their applications.

Meanwhile, mobile devices (e.g. smartphones and tablet PCs) have become an essential element of our lives, providing access to effective communication not limited by time or place. In the past decade, the demand for mobile devices and the expanding growth of mobile Internet traffic have been driving massive improvements in wireless communications and networking [30]. Mobile devices have more computing and communication capabilities than ever before. For example, mobile devices have been embedded with a vast number of hardware and sensors such as cameras, Global Positioning System (GPS) and accelerometers with the higher data transmission rates utilising 4G, 5G and WiFi technologies [31]. Additionally, mobile vehicular terminals such as drones and smart vehicles are developing and improving adding new functionalities. Smart vehicles use features such as cameras, radar, and GPS to collect diverse information from the surrounding environment [32]. Generally, the developments in mobile nodes, including smartphones, drones and smart vehicles, have

made such nodes good platforms for hosting and executing resource-intensive applications. Examples of such applications include augmented reality, virtual reality, self driving vehicular applications [26], road safety applications, traffic efficiency applications, Mobile Crowd Sensing (MCS) [32, 33] and data collection and delivery [34].

Nevertheless, the resources of mobile nodes are still limited and constrained and they cannot satisfy such applications. This is due to the fact that these applications require higher computation and storage resources and have strict delay requirements. Due to current hardware design resulting in limited battery capacity and insufficient computing and storage resources, only fairly uncomplicated computation tasks can be completed using local computing [4, 13, 32, 26, 17, 35, 36, 37]. Such limitations have led to the concept of computational offloading where a mobile device's computing tasks are sent to the Cloud. Running mobile nodes' applications with the support of the Cloud has emerged as an initial solution for enhancing the resources capacity of the mobile nodes'. In the literature, utilising the remote Cloud by mobile nodes is referred to as Mobile Cloud Computing (MCC).

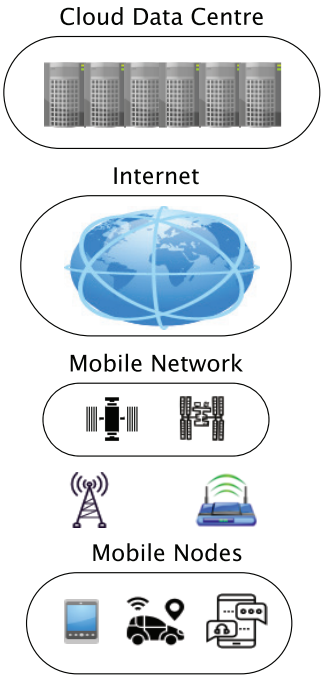


Figure 2.3: MCC architecture.

2.2.1 Mobile Cloud Computing

As shown in the previous section, one of the most critical challenges in mobile computing is to deal with the conflict between the growing complexity of mobile nodes' applications and the limitations of mobile nodes' resources. A feasible idea to deal with such a challenge is to utilise the MCC idea and run the mobile nodes' computing tasks remotely in the

Cloud. MCC is a computing paradigm that utilises the Cloud's resources to deal with the storage and processing issues of mobile nodes' computational tasks. [38] defines the MCC as an integration of Cloud computing technology with mobile devices to make the mobile devices resource-full in terms of computational power, memory, storage, energy, and context awareness. The overall architecture of the MCC is shown in Figure 2.3. In MCC, the mobile node can access Cloud resources using a mobile network (BSs) or access points (Wi-Fi). The mobile node is then connected to the Internet and can access Cloud services [38]. As it can be seen from Figure 2.3, in MCC, the traffic has to pass through several networks including the mobile network (when using 4G for example), backhaul network and Internet which can add extra delay [30]. Using the Wi-Fi for accessing Cloud resource might have less delay as the mobile node will be connected to the Internet without having to go over the mobile network [38]. In contrast, using 4G for example to access Cloud resources might introduce higher delay as this connection has to go over several networks. In both cases, it is going to be higher delay compared to the case when we have the server just one hop away from the mobile node.

MCC was not perfectly capable of handling the mobile nodes applications for several reasons. The potential high number of IP hops between mobile nodes and the Cloud data centres results in longer delay when offloading [30, 39]. Such delay is critical for mobile node applications. Also, the MCC also places a further load on the radio and the backhaul of mobile networks [13]. Additionally, large data centres are mainly focused on serving enterprise users; i.e., high volume workloads, using virtualised resources such as virtual machines. Using the same strategy to serve mobile applications, no matter how small the workload is, will incur an overhead for the overall virtual resources provisioning and management due to the large number of mobile applications to be served [11]. This situation creates a further need to look at the MCC architecture with a view to improvements that would serve the mobile nodes in a better and an efficient way. To deal with this challenge, several studies have proposed new designs to deliver Cloud services to mobile nodes [24, 40, 41]. The common feature between these studies is an architecture designed to deploy small data centres at the Edge of the network.

2.2.2 Edge Computing

As outlined in the previous subsection, utilising large and distant Cloud data centres is not beneficial for enhancing mobile nodes' computational capabilities. To overcome such limitations, the idea of deploying small data centres at the Edge of networks has been introduced, under the umbrella term of Edge computing. Edge computing, as a new form of Cloud computing, brings services close to end users, and thus reduces the response time and minimises the load on the network [42]. The main objectives of Edge computing are high bandwidth,

ultra-low latency, real-time access, and its key characteristics include geographical distribution, mobility support, location awareness, proximity and context-awareness [42].

There are several different models for Edge computing in the literature. [43] summarises three main types of Edge models. In the first model, the Edge servers are deployed to provide computing and storage services to a variety of devices such as sensors and mobile nodes. In this instance, the servers may refer to real servers machine or Raspberry Pi devices as long as these devices provide a minimum computing and storage capacity for other devices. The second approach, the Edge Cluster, uses multiple devices in coordination together to perform certain tasks, as seen in a smart home scenario. The third type of Edge model is the layered approach, where Edge devices are layered based on increasing resource capabilities or by location, as seen in smart cities [43].

Edge computing, in general, has been defined in the literature in varying ways. One of the most common Edge architecture is the MEC system which has become one of the most utilised architecture in the literature as the system is increasingly used for executing new mobile applications [26]. The MEC architecture is intended to be integrated with the mobile networks. This implies more opportunities for mobile nodes for new applications and use cases. This also implies that the MEC will be adopted by the mobile network operators, which will result in faster development of such a concept. As the work in this thesis is mainly designed for mobile nodes at the Edge and so is the MEC, the MEC term will be mainly used throughout the thesis. The following subsections attempt to offer an overview of the Edge terms while narrowing the focus on the term to that considered in this work i.e., MEC.

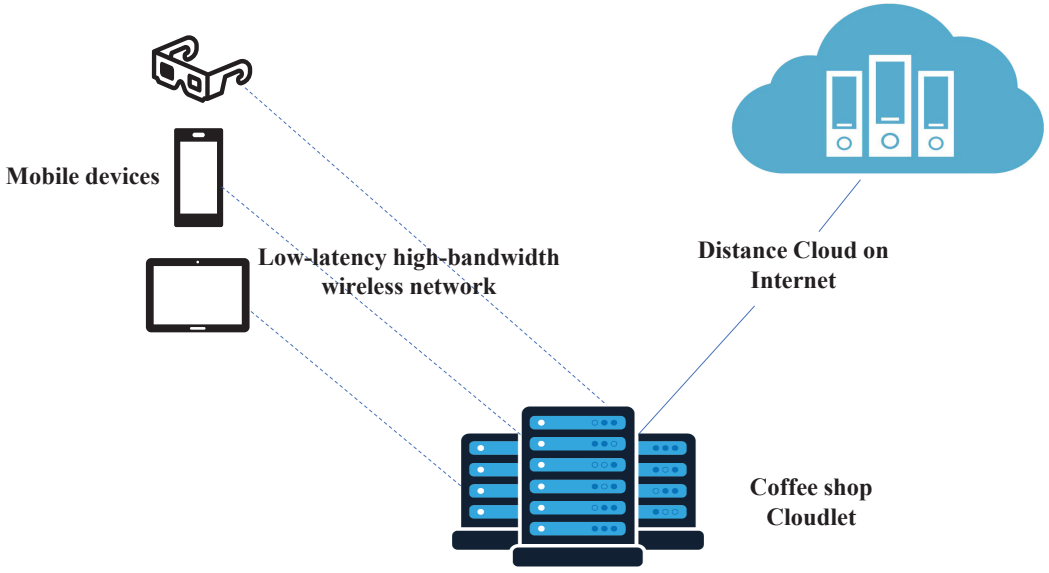


Figure 2.4: Cloudlet concept [24].

2.2.2.1 Cloudlet

Early work which suggested that smartphone users would benefit from Cloud resources at the Edge of their networks referred to the Cloudlet concept [24]. The Cloudlet concept involves local (one wireless hop) computing infrastructure that may be utilised by mobile devices. Cloudlets thus act as decentralised data-centres, and they may be placed at various places, such as coffee shops, to be located closer to the mobile users allowing their compute cycles and storage resources to be leveraged by nearby mobile users. A Cloudlet, which requires a gigabit internal networking and wireless LAN, can be seen as a cluster of multi-core computers [24]. As it can be seen from Figure 2.4, the Cloudlet provides computing resources for resource-constrained mobile devices via wireless LAN which tends to have higher bandwidth and lower latency. The Cloudlet may be monitored and supported by third-party distant data centre on Internet. The deployment of the Cloudlet was envisioned to be just like the deployment of the Wi-Fi access points.

One disadvantage of the Cloudlet, in its initial proposal, is that it is supposed to be accessed via mobile devices through a Wi-Fi connection as the mobile technologies (for instance 3G) at that time were not able to provide the required bandwidth and latency for such a proposal [24]. Each time a user wants to access the Cloudlet, they must thus switch from their mobile network to Wi-Fi to exploit Cloudlet services [13]. The other key disadvantage is that QoS is not guaranteed, as the Cloudlet is not part of a mobile network and a mobile device can easily move out of the coverage of the Cloudlet [13]. Further, the Cloudlet itself is dependent on a robust and uninterrupted Internet connection [39].

2.2.2.2 Fog Computing

Another early architecture proposed for Edge computing was the Fog computing [40]. Fog computing was proposed in 2012 by Cisco to provide a platform for Internet of Things (IoT) and big data applications [40]. In Fog computing, task processing is mainly carried out in the local area network and in an IoT gateway or a Fog node [39]. Fog is most suitable for applications that require low latency as in gaming applications; location awareness, as in augmented reality, geo-distributed and large number of nodes as in sensor networks [40]. Fog computing has some disadvantages due to its dependency on wireless connections, however, as these must be live in order for any processing to be performed. The location of data is also an issue, as data is kept within the local network rather than being distributed on the mobile network [40].

2.2.2.3 Mobile Edge Computing (MEC)

QoS and Quality of Experience (QoE) for mobile nodes are not guaranteed by either Cloudlet or Fog computing, as these architectures are not integrated into the mobile network [13]. Thus, a new Edge computing architecture, Mobile Edge Computing or Multi-Access Edge Computing (MEC), has been proposed to be deployed and integrated with mobile networks. This type of integration allows the service providers to deliver local and fast services from aggregation sites such as BSs [41]. It also allows mobile nodes to access services easily with higher chance of being inside the coverage of services.

In general, MEC, Cloudlet and Fog computing are overlapping terminologies and used interchangeably but they are different in several ways [30, 39]. For example, MEC and Cloudlet provide resources mainly to mobile nodes, whereas Fog computing relies on the hardware designed by Cisco that possesses computational capabilities along with the normal functionality of the device such as router and switches [42]. The MEC term and its related architecture are the main architecture and term used in this thesis, but many aspects discussed are also applicable to Cloudlet and Fog Computing. The next section thus offers further background to the development and use of MEC.

2.3 MEC Overview

MEC overcomes many of the limitations of MCC by bringing Cloud resources (processing and storage) to the Edge within the RAN [39]. MEC, based on the concept of integrating Edge computing into mobile network architecture, was developed in 2014 by the Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI) [41]. The standardisation of MEC was supported by mobile operators such as DOCOMO, Vodafone, and TELECOM Italia, and manufacturers such as IBM, Nokia, Huawei, and Intel [13]. According to ETSI [41], MEC is defined as follows: “Mobile Edge Computing provides an IT service environment and Cloud computing capabilities at the Edge of the mobile network, within the RAN and in close proximity to mobile subscribers”. The main features that distinguish MEC from the Cloud, are listed below [44].

Proximity being close to the source of information makes it easier to capture big data for analytics tasks. This is also beneficial for computation-hungry applications such as augmented reality and video analytics.

Lower Latency as MEC nodes are deployed closer to end users, this considerably reduces latency and provides higher bandwidth.

Location Awareness low-level signalling information for Edge devices can be utilised for local services, allowing the development of new use cases and new services.

2.3.1 Architecture of MEC

A generic architecture of a MEC system is shown in Figure 2.5 [39]. As illustrated, the overall architecture consists of three main layers: mobile nodes, MEC servers and distant Cloud data centres. In the lowest layer, the different types of mobile nodes include smartphones, wearable devices and smart vehicles, all of which can benefit from the resources in the upper layers, represented by the MEC servers. In terms of the second layer, the MEC servers are usually deployed within the mobile network in BS or RSU. The mobile nodes then connect to the MEC server utilising the RAN. At the core of the network are the large data centres that may be utilised by the MEC servers during high volume workloads. The MEC servers are connected to the Cloud via fibre links [6, 45] and they may also be connected to each other, whether by means of a wired network or via X2 link [45].

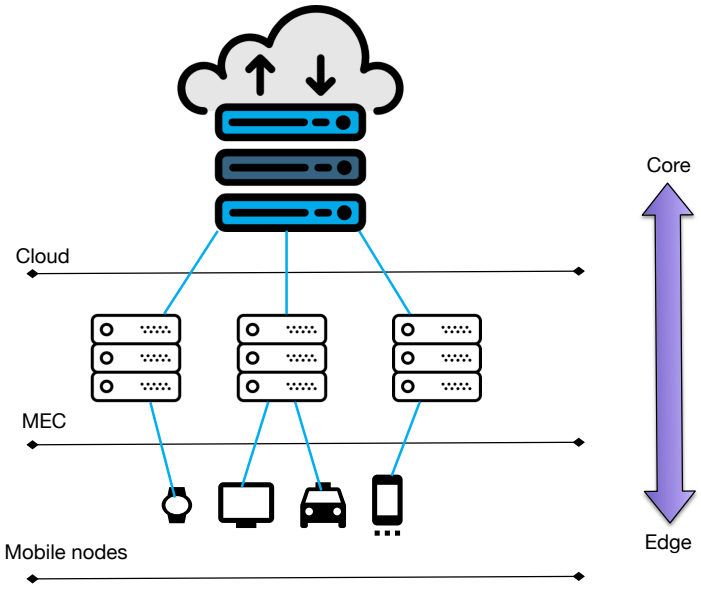


Figure 2.5: MEC architecture.

A detailed reference architecture for the MEC system is provided by ETSI in [46] and visualised in Figure 2.6. It considers the architecture of the MEC system from the position of the provider. According to the proposed framework, MEC will enable the implementation of MEC applications as software-only entities that run on top of a virtualisation infrastructure.

The virtualisation infrastructure is a physical server on which the VMs run and is located in or close to the network Edge [46]. The framework for MEC contains the following entities: system level management, server level management and network level management. These entities have functional elements that handle a user request from the initiation of the request until the termination of the request.

For example, the functional elements in the host include platform, application and virtualisation infrastructure. These elements provide compute, storage, and network resources for

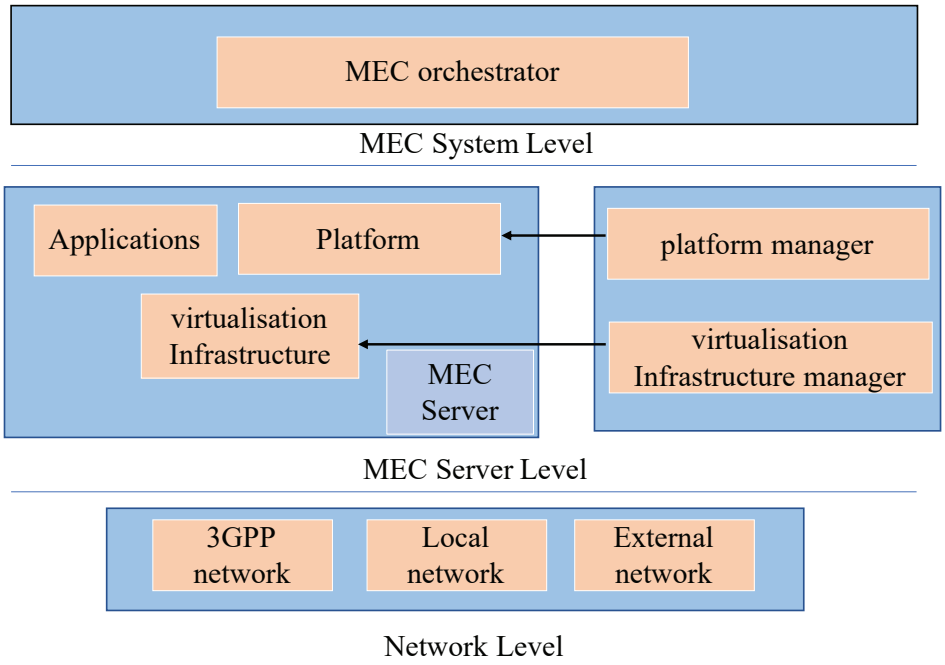


Figure 2.6: MEC framework proposed by ETSI [46] .

the applications. Within the system-level management, the MEC orchestrator is the core function which is responsible for maintaining an overall view of the MEC system based on deployed hosts, available resources, and available services. In MEC host level management, there is a mobile Edge platform and a virtualisation platform managers. The mobile Edge platform deals with the life cycle of the applications, application rules and service authorisation, and traffic rules. The virtualisation platform manager is responsible for the allocation, management and release of the virtualised computation/storage resources provided by the virtualisation infrastructure located within the MEC server.

2.3.2 MEC Servers Deployment Options

There is no specific definition of where an MEC server should be, and the server locations in the system are not specified [30]. Such a problem is named in literature as MEC servers placement, MEC servers deployment or MEC sit selection. However, there are some proposals which suggest where the MEC servers should be deployed. The deployment option depends on the use case [1].

There are many deployment (possible) scenarios for the MEC servers such as in mobile network BSs, or indoor locations such as enterprise buildings (i.e. access points). These deployment options (as discussed in [44]) give the MEC servers the ability to cover and provide different types of services and applications. As mentioned earlier, a use case that can benefit from MEC servers' resources is the technology involved in vehicular networks, where

vehicles equipped with computation communication units offload tasks such as intelligent vehicle control, traffic management, and interactive applications to MEC servers. In such an application, RSU is a candidate location to deploy the Edge servers [5, 6, 9, 20, 32, 45].

2.3.3 MEC Application Scenarios

MEC offers multiple benefits to various stakeholders including the service providers and mobile nodes users [13]. From a service provider perspective, the MEC provides a mean of tools to improve the QoS of the users. An intuitive solution is to utilise the Edge to improve the content delivery caching [39]. For example, as the MEC servers are deployed at the city level, a popular video for a city can be cached in the MEC server providing better QoS and reducing the load on the mobile network. Another example is when using the MEC to provide real-time information about the radio/backhaul network's traffic requirements. Traffic management applications then reroute traffic as required [13], based on the information provided by the MEC.

Mobile nodes in MEC environments can profit from the MEC mainly by computation offloading, which enables the running of new emerging applications [13]. In the following subsections, several use cases for the computation offloading are presented. Generally, these use cases can be divided into two main categories graphical-based applications and data-oriented applications.

2.3.3.1 Graphical Applications

MEC can be used to improve the QoS of users in graphical applications including augmented and virtual reality and video gaming. These applications are considered to be latency-sensitive applications. An example for augmented reality apps is where the mobile node captures an image of a place of interest and shows it on screen [47] or recognises historical monuments (e.g., tourist attractions) and accordingly receives an introductory video about these locations [48]. The results from the performance evaluation in [47] show that the offloading of the augmented reality app from a mobile device to an Edge server deployed closer to the user provides significant reduction of latency up to 88 % and energy consumption can be decreased by up to 93%. The need for powerful and extra resources in this case is proved by the fact that the mobile node resources cannot cope with CPU-hungry applications, as in the majority of circumstances these applications implement machine learning models.

The concept of the ultra-dense network has been researched and proposed as a new MEC-based architecture. It has a large number of low-power small cells with small coverage which are installed inside a single macro base station [49]. Having a number of small cells with computing resources in each of them can improve the capacity of network, computing

resources and network coverage thus enhancing user experience when using augmented reality apps [49, 50]. Due to the shorter distance between the mobile node and the serving cells, it will be an opportunity for the mobile node to exploit multi servers for computation offloading.

Therefore, with multiple MEC servers deployed within small cells closer to the users, computation offloading will be more practical as such deployment is expected to improve the QoE of the mobile nodes when dealing with augmented reality [51]. Meanwhile, due to the large number of potential places for offloading augmented reality apps, the decision of where the task should be sent (or simply which server should be selected) is a problem that needs to be looked at carefully [51].

2.3.3.2 Data Aggregation

Another important use case for the MEC is where the MEC servers act as an intermediary data-processor for large data sensed and collected by mobile nodes for data analysis tasks instead of routing all the data from mobile nodes to the Cloud [52]. The results of the data analysis offloaded to the MEC server are either directed to the mobile node itself or transmitted to other systems. Real-time network data collected by the mobile node can also be utilised by service providers for managing resources such as the number of users in one area.

A potential use case of the MEC servers deployed in RSU deals with the fast evolution of sensory technologies in vehicular networks. In particular, AVs are to be equipped with many sensors such as cameras, LiDARs and radars. Such sensors discern the surroundings to make sure the autonomous system works [53]. It is challenging to process such data due to the huge amount of sensory data and the limited computation ability of onboard computing and storage units. Therefore, MEC servers can be helpful in enhancing the ability of the AV as a place for aggregating different types of data and for further analysis. An outline of the existing work dealing with such a scenario is provided in the following paragraphs.

BEGIN was proposed in [15] as a big data enabled EnerGy-efficient vehicular Edge computing. BEGIN mainly consists of two domains: the computing domain and the big data domain. The computing domain consists of a Cloud layer, which has a SDN controller and the Edge layer. The big data collected and processed in the data domain are used to provide energy efficient Edge computing. RedEdge [54] is another big data processing architecture that incorporates a mechanism which facilitates the processing of big data streams at the Edge near to the user. The RedEdge considers mobile devices to be a main platform for data processing. However, in the case of the unavailability of computational and battery power resources, it offloads data streams to local mobile Edge devices or to the Cloud. In RedEdge, however, the Edge nodes refer to a set of mobile devices which are connected in an ad hoc

network. The idea of having the MEC servers as a platform for data analytics was also proposed in [55]. The proposed MEC architecture is for a MCS service. The MCS refers to a human-driven IoT service to which people send their observations of different phenomena in their surroundings by sharing their sensor data while on the move. The authors proposed an MEC architecture for the MCS by distributing MCS tasks to multiple MEC servers deployed at the Edge of the network [55]. The work in [56] proposed a decentralised traffic management system to minimise the average response time caused by traditional centralised traffic management. The system contains a Cloud layer, a Cloudlet layer, and a Fog layer. The Cloud layer is always far from vehicles and formed by trust third authorities. The Cloudlet is a server which can be installed in each region of the city and can be accessed via the RSU. Fog nodes are formed by vehicles (parked and moving) within the communication ranges of RSUs. In short, the scenario in this work is that vehicles can upload sensed events (e.g., traffic jams, car accidents, and road surface damages) to a nearby RSU. When an RSU receives a message, it will send the message to Cloudlet or Fog nodes for processing. Then, the extracted information will be uploaded to traffic management system for further actions.

The work in [57] puts forward a strategy of computation offloading for a specific data mining application, namely, activity recognition for mobile devices. As the user moves, data is obtained from various places and is held in storage on the mobile device. This data is examined so that a choice can be made as to whether to offload to an Edge server or the Cloud, or to carry out the process on the device. Should it be decided that offloading to an Edge server takes place, the device's communication interface obtains a list of Edge servers and connects to the best server. However, during the movement of the mobile device, e.g., in AVs use case, a better server could be present and accessible, which is however not picked up by the communication interface whilst the device's communication interface scanning. Therefore, with regards to execution delay, it is possible that a better MEC server is available as the mobile node moves. This opportunity is taken into consideration in the proposed schemes in this thesis. Another use case involving data offloading is the case where Unmanned Aerial Vehicles (UAVs) collects data for different systems and offload it to an Edge server for third party. For example, the use case considered in the work [58].

To sum up, the studies described above do not address the site and time selection problem in task offloading. They were primarily concerned with whether a task should be performed locally or offloaded. Moreover, the task offloading decision in these proposals is based on complete information gathered by either the mobile node as in [57] or by centralised controller as in [15, 54]. More importantly, the work presented in this thesis can be used to support such use cases. It supports these scenarios by optimising how the mobile node chooses an Edge server (or time) for data offloading. For example, it helps when many Edge servers are available for offloading. When one Edge server is available, the mobile node can observe the offloading control variable, e.g., the processing time, in a time-slotted manner,

where the time is divided into equal small intervals and each interval represents one decision slot. Table 2.1 provides a summary of these studies and how the OST-based models can be used in such use cases.

Application scenario	Motivation for an OST-based model
Ultra-dense network [49, 50] where multiple MEC servers deployed within small cells closer to the users.	Due to the large number of potential places for offloading augmented reality apps, the decision of where the task should be sent (or simply which server should be selected) is a problem that needs to be looked at carefully [51].
Using big data analytics to provide efficient, programmable, scalable, and flexible framework to manage vehicular edge computing, e.g., BEGIN [15]. Employing the Edge node for big data processing offloaded by Edge devices as in RedEdge [54] and MCS [55].	In this type of application, one important source of data is mobile nodes such as vehicle sensors. Thus, the proposed models can be utilised in this use case to optimise the way mobile nodes offload the data to an Edge servers, e.g., time and server selection taking into account the data quality, e.g., the timeliness of data.
Task offloading for data mining applications such as activity recognition task as in [57].	The proposed models can be used in this use case to provide server and time selection decision-making in sequential manner without having to gather all the servers information, i.e., make a decision based on incomplete information.

Table 2.1: A summary of some MEC applications and the motivation for using an OST-based model when offloading.

2.4 Computation Offloading

Computation offloading generally refers to sending resource-intensive computing tasks to an external server in order to enhance the efficiency of a mobile node's application. In line with the ongoing development of Cloud and Edge computing paradigms, computation offloading has been adopted as a solution to augment the computing resources of mobile devices. The underlying concept that developed into current forms of computation offloading was that of Cyber Foraging, as proposed by Satyanarayanan in [59]. This involves dynamically enhancing the computing resources available to a wireless mobile node by exploiting wired hardware infrastructure. This idea has received much wider attention with the introduction of Cloud computing by Amazon and Google and the advances in mobile nodes driven by

iPhones and Android devices in the mid-2000s. These developments led to the idea of utilising Cloud resources to support mobile devices, leading to the creation of MCC as described previously.

The idea of computation offloading was further supported by the work done by Satyanarayanan in [24] which introduced the concept of the Cloudlet. There are generally two main approaches towards remote execution [25]. The first one occurs when the application programmer defines which parts of an application should be remotely executed based on the resources required by the relevant component. The second approach is when the entire application is offloaded in the form of a virtual machine or container. Further background about the computation offloading concept, followed by a review of key work relating to this concept is offered below.

2.4.1 Offloading Process

Computation offloading consists of a transmission phase, a remote execution phase, and results being sent to the mobile node [14]. The offloading computational time for an offloaded task is therefore measured as the sum of the time required to send the data to the MEC servers, the time it takes to process the data at the relevant MEC server, and the time required to send the results of the computations back to the mobile node. Given these stages, there are several different random variables involved in the process that may be used as control variables or decision variables.

The process inside the mobile node starts by deciding either to offload or do the task locally. This might involve decision-making algorithm to optimise such a decision. If offloading the task beneficial for the mobile node, the next step is to look for the offloading site where the task will be executed. Depending on the available options, the mobile node selects the appropriate site for offloading. The work presented in this thesis deals the decision of selecting an appropriate site for offloading mainly in multi Edge servers scenarios. Figure 2.7 shows the decision-making process done by the mobile node.

2.4.2 Offloading Metrics

Several previous studies have utilised a range of metrics to examine overall performance in computation offloading [26]. Such metrics include energy, latency and cost. They either have been considered together and separately.

Energy When offloading, a mobile node consumes energy in order both to transmit the data and to receive the data from the Edge server. The transmission power, the size of data packets

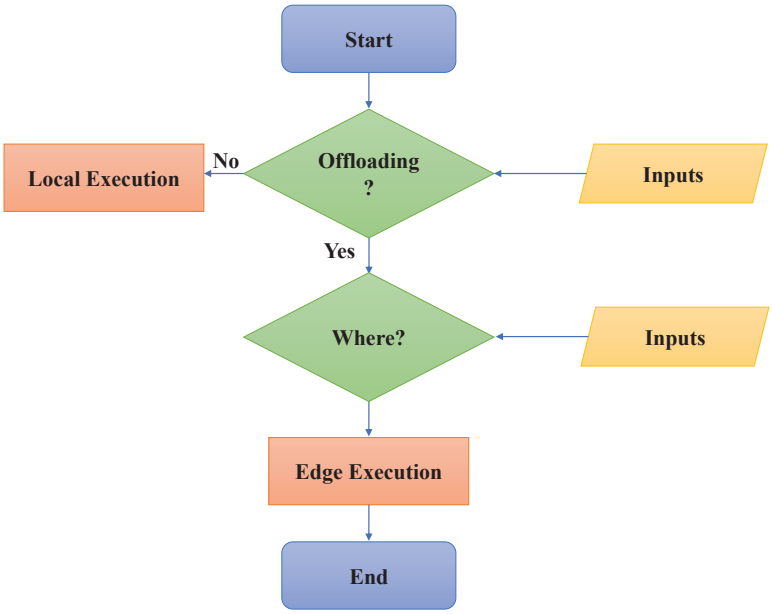


Figure 2.7: Process of offloading in the mobile node [26, 38].

transmitted and received, and the bandwidth of the wireless channel are thus the main factors affecting the energy consumption for computation offloading.

Delay The delay in computation offloading is measured as the total time required to send the task and the related data to the Edge server added to the execution time in the server and the time required for results to be returned from the Edge server. The bandwidth of the wireless channel and the processing time, including the overall load on the MEC server, all affect such delay.

Cost Some studies within the literature have considered the fact that most of these metrics can be represented by cost. This might refer to the cost of transmission time or processing time in the Edge server, for example [26]. Although many factors play a big role in overall system performance, the focus of this thesis is on processing time in the MEC server.

2.4.3 Decision-Making in Computation Offloading

Several different types of decisions must be taken during the process of computation offloading, and many of these have been considered in the literature [43, 14]. The decision-making is not limited to the decision of performing tasks locally at the mobile node or offloading them to an Edge server, but also includes the decision-making of the other possible actions involved during the process of offloading. The survey in [14] identified the actions that require a decision-making during the offloading process, which are summarised in the following page.

Executing Locally or Offloading The mobile node first needs to decide if it is worth it offloading a given task to external server.

Server Selection If the result of the previous decision is to offload, then the selection of an appropriate Edge server is important when multiple servers are available.

Wireless Resource Allocation The quantity of wireless resources, including the frequency that should be allocated for offloading, must be determined.

Power Setting The appropriate transmission power level should be set for offloading.

Computation Resource Allocation Computation resources must be allocated to the task in the form of CPU cycles or a virtual machine.

In this thesis, the decision of selecting an appropriate server when multiple MEC servers are available is considered. Such a consideration can be also generalised to the decision of selecting an appropriate time for offloading. The selection of a server for offloading generally occurs in scenarios of one to many offloading [14]. In this case, many Edge servers are available for the mobile for offloading. In addition to the decision of whether to offload, the mobile node must also decide which server the task should be offloaded to.

2.4.4 Early Frameworks for Computation Offloading

Early frameworks for computation offloading include MAUI [25], CloneCloud [60] and ThinkAir [61]. These frameworks were proposed between 2010 and 2012 when the Cloud was the main attractive means to provide the necessary resources for mobile nodes. In addition, the frameworks were proposed in conjunction with the initial proposal of the Cloudlet concept, which was discussed in Section 2.4. These studies cover important design features as well as key findings that were the foundation for most of the task offloading research. A review of these frameworks is provided in the following paragraphs.

MAUI's main objective is to minimise the power consumption considering the latency constrains, and it allows developers to produce an initial partitioning of their applications and thus to define what parts of those applications should be offloaded. In the smartphone side, the MAUI has an interface to the decision engine, known as a proxy, which handles control and data transfer for offloaded methods and a profiler, which instruments the program and collects measurements of the program's energy and data transfer requirements. On the server side, MAUI has four components: the profiler and the server side proxy, the decision engine, and the MAUI coordinator, which handles authentication and resource allocation for incoming requests.

Determining which methods should be executed locally or executed remotely is done by the MAUI solver. This problem is formulated as integer linear programming problem. To

save energy, the optimisation solver runs on the MAUI server side and not in the mobile node. The work was evaluated under different connectivity types including Wif-Fi and 3G with different Round Trip Time (RTT). The results showed that the less the RTT, the better the performance the mobile node gets in terms of the power consumption and the execution time. Important aspects related to the remote execution such as the portability of the code are those considered in this work. This work does not consider the state of the external server to be used for the offloading decision; instead, only the device and the network characteristics are used to drive the offloading decision. This work, i.e., MAUI, supports that idea that offloading to distant data centres (Cloud) is not a good idea, suggesting that remote execution should be done by nearby servers (Edge), such as those proposed in the Cloudlet concept.

ThinkAir [61] is a framework that helps developer migrate mobile device applications to the Cloud. Setting aside the aims of MAUI, the authors of ThinkAir argued that, due to increases in the connectivity capabilities of mobile nodes with less round trip time, the Cloud offers a more attractive solution for the mobile node constraints. ThinkAir consists of three components: the execution environment (mobile node side), the application server (server side) and profilers. An execution controller decides whether to offload execution of a particular method or to allow it to continue locally on the phone based on data collected by the profilers about the current environment. Beginning with the development stage of a mobile application, the programmers can thus define which processes need to be offloaded to the Cloud. The main goal of ThinkAir is to make decisions about which parts of the application should be offloaded. ThinkAir was evaluated under different connectivity scenarios utilising different types of mobile applications. The results demonstrate the benefits of ThinkAir with the ability of on-demand VM resource scaling and exploiting parallelism. ThinkAir focuses more on scalability issues and parallel execution of offloaded tasks in the Cloud.

CloneCloud [60] is another offloading framework that aims to automatically determine which parts of an application should be sent to the Cloud. The relevant high level architecture includes profilers on both sides, a partition analyser in the Clone VM side, and a Manager on each side for offloading process management. The partitioning of an application operates utilising three components: a static analyser, a dynamic profiler and an optimisation solver. The partition aims to minimise an objective function based on the overall cost of each operation (computation cost and migration cost). The output of this optimisation is then assigned to binary decision variables for every process in the application to determine whether it should be run locally or remotely.

These frameworks have defined the fundamentals of computation offloading in terms of the components that should be deployed in mobile nodes and in the server side, e.g., the Cloud. They also focus on the technical aspects of mobile applications, such portability and the partitioning of applications. Further, their focus on making decisions as to whether there is a need for offloading or whether execution should actually be performed locally.

The concept of network profiling is also another important point discussed in these studies that can be used in any offloading decision-making in the mobile node. In MAUI, for example, each time MAUI offloads a method, the profiler obtains a more recent estimate of the network characteristics such as latency and bandwidth. The same idea was used in ThinkAir and CloneCloud. Such a concept can be also extended to gather other information such as the server load or the execution time. This information can be then used for decision-making models deployed in mobile nodes. In the proposed work in this thesis, the mobile node is required to observe the current status of the random variable to be optimised; thus, the proposed decision-making can be combined with computing and network profiling concept to offer the present state (value) of the observed random variable.

2.4.5 Summary of Offloading Decision Work

Building on the early frameworks, recent and various pieces of research have been carried out to deal with the issues of offloading data and computing tasks to an Edge node. The majority of which have emphasised if there should be local processing of the data or task, or whether it should be offloaded externally, for example, to an Edge server or to a public Cloud.

In the following paragraphs, an outline of the recent research work is presented and compared to the work presented in this thesis. We start by discussing recent papers with different approaches then focusing on the common method that has been used extensively in the computational offloading, i.e, reinforcement learning.

The work in [62], for the purpose of reducing the latency of task execution as well as energy consumption, presents the idea of collaborative MEC. This work focuses on UAV applications that uses photos and videos for tasks like object identification or obtaining traffic information. The captured photos/videos are then offloaded to an Edge server. When the task is generated by the UAV, a system orchestrator should determine which server should be selected, what data rate ought to be adopted to transmit data to the selected server and how much workload each server (cooperator) should be allocated. This work [62] is based on the assumption that the system orchestrator makes this choice. However, in our work, the decision is autonomously made by the mobile node itself as in some situations, there might be heterogeneous or different operators for the MEC servers.

A context-sensitive offloading system using machine learning classification algorithms was proposed in [63]. The proposed system integrates middleware, machine learning classification algorithms, and a robust profiling system. The authors considered whether a task should be done locally or at the Edge node. Our proposed work can help such a system to decide which server to be used and what time the offloading should occur once the decision is made by such algorithms.

The work in [36] presents an intelligent computation offloading based on Mobile-Edge Cloud computing architecture. The considered architecture consists of three layers: mobile nodes, Edge servers and a remote Cloud. Optimised task offloading and migration rules based on a prediction of tasks' features using Long Short Term Memory are presented (LSTM). In this work, three approaches can be used for tasks processing: locally in the mobile devices, in an Edge server, partially offloading (local and external at the Edge server). The task's feature to be predicted in this study is the data size for the task because the total delay of computation offloading is related to data size as argued by the authors. Thus, the objective is to minimise the total delay given the predicted data size. Similarly, for the migration process, the objective is to minimise the migration delay based on the data size prediction for the task to be migrated. The authors did not explicitly define where the prediction model will be executed as the model needs some training based on historical data, which might be resources usage cost. Also, the migration prediction-based was not utilised in the experiment to show the effectiveness of the proposed model. More importantly, the offloading decision here is about whether a task should be offloaded or executed locally in the mobile device, which is different from the sequential decision-making algorithm proposed in this thesis.

In [37], an autonomous computation offloading strategy in MEC focusing on a multiple-user multiple-server environment with heterogeneous services and Edge-Cloud platforms is proposed. The authors utilise the concept of Monitoring, Analysing, Planning and Executing (MAPE-k) loop to take a decision of whether the task should be executed locally or offload to an external server. The objectives are to minimise the execution delay and power consumption. Deep Neural Network (DNN) is used to solve the optimisation problem. Also, a Hidden Markov Model (HMM)-based model is used to select the transmission media when offloading. While the study adopted powerful algorithms for the decision-making process, the proposed framework may introduce an overhead during the decision-making process and require more resources to be implemented. In addition, the mobile node is dependent on other nodes on which some components of the framework will be deployed.

The work in [64] proposed an offloading decision algorithm for vehicles. The proposed algorithm decides which part of the application should be done locally or in the Cloud-based on the task requirements. A heuristic mechanism for partitioning and scheduling the application between the vehicular and the Cloud is proposed. This work is designed for Cloud-based architecture and focused on the decision regarding which part of the application should be offloaded.

The work in [65, 66] investigated a multi-sites offloading decision based on Analytical Hierarchy Process (AHP) multi-criteria method. The multi-sites include the mobile device, near by mobile devices, Cloudlet (Edge server) and too far Cloud server. This study also contributed with the design and development of an Android offloading enabled framework that can be adopted by developers to build MEC applications. The assumption in this work is that

the mobile devices will have to collect the offloading devices' information and based on such information the decision is made. Different from this work, in our work, the mobile node is not required to collect all the Edge devices' information. In other words, in our setting, the mobile node is observing the Edge nodes (sequentially) and does not need all the Edge servers' information in advance for the decision-making.

The work in [20] considered the computation offloading in Vehicular Fog Network (VeFN). The authors provide a review of the offloading decisions work in VeFN. According to the authors, the offloading decision in the VeFN can be classified into three major modes: vehicle to vehicle, vehicle to RSU and pedestrian to vehicle offloading. This work provided two use cases: the first one is learning-based task offloading applying Multi-armed bandit (MAB) focusing on vehicle to vehicle offloading. The other use case is a delay-constrained task replication exploiting vehicle mobility where RSU collects task by vehicle and pedestrian and offload it to nearby Fog vehicle. An interesting point considered by the authors in this work is that mobility is not always an obstacle and can be a supportive factor to help in finding better resources for offloading. This is an important aspect we build our model on where we allow the mobile node to explore the MEC servers with the objective of finding a better resource.

The work in [67] proposed a distributed and context-aware task assignment algorithm in MEC environment. The task assignment in this work refers to the decision of where a task should be offloaded. The problem is formulated as an one-to-many matching problem by taking into account the devices' and MEC servers' computation capabilities, wireless channel conditions, and delay constraints. The main objective of this work is to reduce the overall energy consumption while satisfying task owners' heterogeneous delay requirements. The proposed work is compared with the Random matching scheme where tasks and Edge nodes are randomly paired together, the scheme where the Edge nodes with higher computational capability has a higher priority to accept tasks and the centralised method, i.e., a centralised authority with complete information searches through all possible combinations to find the optimum solution. The proposed solution was the closest one to the centralised method in terms of energy consumption and the average utility. Even though the authors' main idea in this work is to make a distributed tasks assignments, still, the nodes including the Edge nodes and the mobile nodes are required to collect information from all the neighbors devices before making the decision of offloading which is different from our work, where the mobile node proceeds sequentially for decision-making without having to know about all the Edge nodes in advance.

Similarly, matching problem was also utilised in [68] where MEC system with multiple servers and mobile devices (users) is considered. Each user comes with a computation task that needs to be offloaded to a MEC server for execution. The task offloading part adopts the method of matching theory and the transmit power allocation part adopts a heuristic

idea. In particular, the work proposes sub-optimal algorithm for joint task offloading and transmit power allocation. The task offloading is divided into two sub-problems: server allocation and channel allocation. These two sub-problems are being modelled as user-resource one to one matching problem. The work is evaluated in terms of system delay and energy expenditure and compared with a many-to-one matching and a one-to-one matching, the nearest MEC server and the random approach where the task offloading decision of each user is chosen randomly. The experimental results show that the proposed model cannot only obtain less delay, but also generate less energy consumption when the data amount of tasks is the same but the workload is random in the MEC system. The proposed algorithm requires the information of the MEC servers and the users around these servers as inputs which makes it different from the method proposed in this thesis. In Table 2.2, a summary of the work discussed above is provided. The main difference between the work proposed in this thesis and these studies lies in the type of decision that was taken into account (mainly executing locally or offloading) and the requirements for a global view of the Edge servers.

Papers	**Key difference**
[62]	This work is based on the assumption that the system orchestrator (control station) makes the offloading decision.
[63]	The authors considered whether a task should be done locally or at the Edge node.
[36]	The authors considered whether a task should be done locally or at the Edge node.
[37]	The mobile node is dependent on other nodes in which some components of the framework will be deployed.
[64]	Focused on the decision regarding which part of the application should be offloaded.
[65, 66]	The mobile device is required to have the offloading sites' information.
[20]	Focused on the benefits of utilising the mobility using online learning and task replication.
[67]	Mobile nodes are required to collect information from all the neighbors devices before making the decision of offloading.
[68]	The proposed algorithm requires the information of the MEC servers and the users around these servers as inputs.

Table 2.2: A summary of related work.

2.4.5.1 Reinforcement Learning Based Solutions

In general, reinforcement learning is concerned with how an agent acts in an environment in order to maximise a cumulative reward. In reinforcement learning, the agent learns by interacting with the environment. The decision-making process is then taken through the feedback received from such an environment. A reinforcement learning model includes a state, an action, and a reward for each action. Examples of the reinforcement learning techniques are Q-learning, Deep Q-Learning (DQL) and Deep Q-Network (DQN) [26]. Reinforcement learning is a common method that has been used to deal with decision-making problem in computation offloading [26]. The recent studies that have used reinforcement learning techniques in computation offloading decision-making are reviewed in the following paragraphs.

A spatial and temporal computation offloading decision algorithm, ST-CODA [69], is related to the work presented in this thesis. This work considers the decision-making of the mobile device in terms of the time and location for offloading tasks by considering the computation nodes and the transmission costs in Edge Cloud-enabled heterogeneous network. Our work's objective and policy differ from ST-CODA because the time-optimised sequential decision only offloads tasks to the Edge servers and not further to the Cloud. In [69], the temporal decision refers to deferring the offloading decision until a low cost network is found, e.g., Wi-Fi network. In our approach, we defer the offloading decision until a lightly loaded server is founded.

The authors in [70] proposed a code offloading framework for offloading in a mobile Fog environment. The proposed method determines which part of the application should be offloaded and takes an offloading decision considering the current state of the Edge node resource by modelling the problem as Markov Decision Processes (MDP) and training it using the Q-learning approach [71]. Also, their algorithm supports the mobility of the user by migrating the offloaded part from one node to another. Their main objective was to minimise the delay of the offloaded applications. In this work, the feasible sites for offloading are: the mobile Fog in close proximity, the adjacent mobile Fog, or the remote public Cloud. In our OST-based approach, we assume that the mobile node can only offload to an Edge server (potentially the most appropriate one with high probability) and there are a set of feasible MEC servers to offload.

The authors in [72] consider offloading decision and resource allocation in the MEC environment by applying a Reinforcement-Learning-based State-Action-Reward-State-Action (RL-SARSA) algorithm. The main goals are to balance the processing delay and the energy consumption when offloading, to define where to offload the task, and to provide efficient resource allocation in the MEC servers. This study takes the advantage of adjacent Edge servers as well as remote execution to improve the decision of where to offload the task. In particular, four sites are considered for offloading the task: local execution, nearest Edge

server, adjacent Edge server, and remote execution in the Cloud. The proposed RL-SARSA was compared with the Reinforcement Learning-based Q-Learning (RL-QL) and the results show the superiority of the RL-SARSA over RL-QL. The limitation of this work is that the proposed model is more centralised and it is executed through an Edge computing controller in each region as indicated by the system model within the paper. This is different from the proposed approach in this thesis where we try to make the mobile nodes more dependent and run the decision-making algorithm locally at the mobile nodes.

The authors in [73] proposed an adaptive computation offloading resource allocation strategy in MEC environment. The objectives of this work are to decide whether a task needs to be offloaded or executed locally in the mobile node. Also, the algorithm helps deciding which node should be selected. The proposed algorithm uses the Deep Reinforcement Learning (DRL) based on DQL. The results of this work show that the proposed algorithm outperforms the traditional Q-learning, the weighted round robin algorithm, the Random and when executing the task locally in different scenarios including different data sizes, different numbers of mobile node and base stations. There is no much information about the experiment platform. One concern arises here is the space and time complexity of the proposed algorithm as there was no analysis about this aspect.

Similarly, the work in [45] employed DRL in a MEC environment for joint server selection, cooperative offloading and handover problem. The contributions of this work are collaborative offloading mechanism between MEC servers and centralised Cloud, a Q-learning algorithm as a baseline solution, and DQL-based algorithm for the considered problem. The possible decisions on the considered setting are executing the tasks locally or at the MEC server, forwarding the tasks to another BS, handover the user to other MEC server, or offload tasks to the Cloud. If the offloaded task cannot be executed by the selected MEC server, the BS might forward (or handover) the task to other BS. The authors in this work tried to overcome the limitation of Q-learning, which cannot deal with high-dimensional and continuous state and action spaces, for obtaining the optimal policy of the modelled problem by adopting a DQN. The work is evaluated using simulation approach and it is compared with a Q-learning-based algorithm, server (optimal) and local computation. The performance metrics include delay, task success rate, energy consumption and ratio of offload tasks. In general, the proposed model was the closest one to the server computation method where all the users' tasks are offloaded to the MEC servers or the Cloud. The assumption in this work is that the MEC servers belong to the same network operator so that the computational data can be split and forwarded among the MEC servers for collaborative execution. Moreover, the proposed model is to be implemented on each MEC server with collaborative model in sharing network of computation resource with other MEC servers or on each MEC server. In other words, it is different from the work presented in this thesis as the mobile node is dependent on other nodes, i.e. MEC servers and network controller.

The paper in [19] utilises the DQL method for optimising the offloading system in MEC-enabled vehicular network in an urban area. The main goal is to determine the optimal target server and the transmission mode selection schemes in order to maximise the utilities of offloading under given delay constrained. Also, to cope with transmission failure in vehicular networks, an efficient redundant offloading algorithm, which also ensures offloading reliability while improving the gained utilities, has been proposed. The problem is transferred into a utilisation function with the aim of maximising the difference between the maximum delay tolerance of a task and the total time cost for offloading task. The system is modelled as a MDP and the optimal offloading strategies are obtained utilising DQL method. The paper also presents redundant offloading schemes to achieve reliable transmission between the mobile node and the target MEC server when utilising V2V and vehicle to RSU. The work is evaluated using real data sets for taxi movements. The work was compared against best transmission path, best MEC server, a greedy algorithm, and a game theoretic approach. The proposed model has the highest offloading utility. This work assumes that the proposed algorithms are to be implemented in a control center which collects the state information from the mobile nodes in the road through cellular networks.

The OST was adopted in [16] for the objective of deriving a good balance between the gain of choosing the best Edge device and the accumulated cost of deep resource probing. The authors in [16] try to enhance the ability of an OST-based model by utilising layered learning mechanism to define the OST thresholds and the sequence of the Edge nodes used for offloading. However, such enhancement results in significant computational overhead and battery consumption for the mobile nodes as it implements DNN and DQN. Also, in their applications, the assumption is that the mobile node will have a list of Edge devices once a task is generated and then the mobile node will define which Edge node makes a good balance between the cost of probing and the execution delay of the task in advance.

Table 2.3 highlights the key differences between the proposed methods in this thesis and the reinforcement learning-based solutions in task offloading decisions. As reinforcement learning-based solutions, in general, require more computing resources for their execution; we note that, in some of these studies, mobile nodes remain dependent on external servers or control in the decision-making process.

Papers	Key difference
[69]	This research investigates whether to postpone task processing in order to find a low-cost network, e.g., Wi-Fi, to offload to the Edge, process the task by itself, or offload the task to an external Cloud.
[70]	This work looks at how to deploy some blocks of the applications in a mobile Fog node in close proximity, an adjacent mobile Fog, or a remote public Cloud to support parallelism.
[72]	The proposed work is a centralised method where and the selection for an offloading site is executed through an Edge computing controller in each region.
[73]	The proposed algorithm decides whether a task needs to be offloaded or executed locally in the mobile node with a selection of an Edge node.
[45]	The proposed solution is to be implemented on each MEC server with collaborative model. As a result, the mobile node is dependent on other nodes, i.e., MEC servers and network controller.
[19]	The proposed algorithms will be implemented in a control centre that gathers state information from mobile nodes on the road via cellular networks, i.e., the mobile node is dependent on the control centre on the decision-making process.
[16]	Built on the assumption that the mobile node will have a list of Edge devices once a task is generated.

Table 2.3: A summary of reinforcement learning-based solutions.

2.4.6 MEC Server Selection

Deciding which server to use for offload has been first considered in the MCC environments. For example, the work in [74] considers the problem of energy minimisation for a MCC system. In this work, the MCC system consists of a group of mobile devices and a set of

servers in the data center. Each mobile device runs an application and tries to upload a portion of its application to one of the servers. The offloading strategy involves two decisions, (1) the amount of computation to be offloaded, and (2) the destination of offloading. In particular, given a number of mobile devices and a number of servers, for each device, find a server for computation offloading, such that the overall MCC system energy is minimised while the performance constraint is satisfied. The problem is modelled as a game theory where each mobile device is a player and his strategy is to select one of the available servers. The proposed algorithm finds the Nash equilibrium for the energy minimisation problem defined above. Nash equilibrium is a concept used in game theory and refers to the state in which no player can benefit by changing its strategy. The proposed model was compared against the Random where randomly assigns the mobile devices to a Cloud server. The proposed Nash-overall policy could achieve large energy savings compared to the Random policy. The proposed algorithm is for MCC architecture and is to be executed in the dedicated server, e.g., the utilised virtual machine.

The authors in [5] considered an architecture where the MEC servers are deployed at the Edge of networks with the support of RSUs to provide services to passing vehicles. Vehicles may pass by several RSUs with their connected MEC servers during the task/data offloading process. These vehicles can offload their computational tasks to any MEC server they can access. The authors introduced a predictive off-loading framework in vehicular networks. Two communication methods relaying via the dedicated short-range communication (DSRC) were proposed: Vehicle to Infrastructure (V2I) and Vehicle to Vehicle (V2V). For the first method, when a vehicle generates a task, the vehicle sends the required data through the roadside unit, and obtains the results from another predicted roadside unit, which will be near the user when the result is ready. For the second method, when a task is generated, a vehicle sends the required data through other vehicles on the road. The data is submitted to the roadside unit, which the user is more likely to connect to when the result is ready. For the first method, the task is always submitted to the first MEC. In our work, we delayed the decision in light of connecting to a better MEC server by applying the concept of OST. Our work fits such a scenario and can be implemented in the mobile node (i.e. the vehicles) to decide which server should be selected for the offloading and when the offloading should start considering the existing offloading framework.

The work in [17] proposes a framework of task offloading for MEC utilising SDN in ultra dense network. Ultra-dense network is proposed for 5G to cope with the high demand on wireless. Ultra-dense network includes small cell BSs and macro cell BSs. The Edge Clouds are equipped at each BS. This work proposes deploying controller at macro cell BS to have a global information about the mobile devices, base stations and MEC servers load. Thus, the mobile node is advised for the optimal offloading decision. The paper focuses on the decisions of (1) whether the task should be offloaded or executed locally, (2) which Edge

Cloud should be used for offloading and (3) decide how much computing resource of Edge cloud should be allocated to each task. These problems are formulated as a mixed integer non-linear program. The authors then transform this optimisation problem into two sub-problems, i.e., task placement sub-problem and resource allocation sub-problem. This work is an example of the centralised method where the decision-making of selecting an Edge server for offloading is made by another node (i.e. SDN controller in this case) which is different from the work presented in this thesis.

The authors in [6] proposed a decentralised management scheme for Edge servers and an offloading approach in the MEC environment. The proposed idea is based on the Peer to Peer (P2P) networking architecture where peers (MEC servers and moving vehicles) have equal privileges. The idea is based on Edgecoin virtual currency where MEC servers and vehicles can store the entire history of the Edgecoin transactions (every transaction by every vehicle), and every workload update transaction by every online MEC server. Based on the previous architecture, two algorithms were proposed. The first one is for the generation of all candidate MEC server(s) in the vehicle moving direction. The output of the first algorithm is the set of MEC servers to be utilised based on their service ranges. An R-tree was constructed to generate the set of MEC servers that are good enough to be used by the moving vehicle. The second algorithm is for determining the optimal MEC server from the generated list to be used by the moving vehicle. Different from this study, in our work, we are trying to make the mobile node more independent with respect to the offloading decision-making. The work requires the mobile node (vehicle) to be involved in P2P network which might not be available for the mobile node all the time.

The authors in [75] proposed a framework for joint network and virtual machine selection in a Cloudlet environment. The authors of this work considered the Cloudlet architecture and the QoS of a face recognition application as an input for the proposed system. While the user is trying to offload in corporate and campus networks, many Wi-Fi access points might be available during the offloading sessions. Thus, the objective is to select the appropriate network along with VM resources that guarantee the quality of access to the Cloudlet resources. This work assumes that all access points with their information are available to the mobile node, which is a different setting from ours. The main differences between the proposed methods in this thesis and server selection methods from the literature are summarised in Table 2.4. While these studies have mostly focused on the decision of which Edge server to use for offloading, it is important to note that the mobile node is unable to make this decision on its own.

Papers	Key difference
[74]	Designed mainly for Cloud environments.
[5]	Focused on predictive off-loading framework to predict which MEC server the mobile node will be used to receive the results of tasks when uing V2V communication methods.
[17]	The decision-making process is made by an external node, SDN controller in this work.
[6]	It requires the mobile node to involve in P2P network.
[75]	Complete information about the Edge servers is required to make the selection.

Table 2.4: A summary of server selection proposals.

2.4.7 Mobility Management

In the earlier mobile cellular system, the mobility of nodes is managed by handover operation when users change the connected cell as nodes roam between the cells to guarantee the service continuity and QoS [13]. Similarly, in a MEC environment, there is a need for keeping the continuity of the provided service. In such a situation, there can be two types of mobility: low and high mobility [13]. An example for the low mobility is when the user is roaming in a building or a campus. Vehicular networks technology, on the other hand, is an example of higher mobility.

2.4.7.1 Low Mobility

Regarding low mobility, previous work has considered the power control in the Small Cell Cloud (SCC) setting when offloading a computing task as in [76], [77]. The basic concept behind this approach is to add computational and storage capacities to small cells like microcells, picocells, and femtocells [77]. In such a setting, for some applications and service, there is a delay constrain in which the request has to be satisfied by the current connected cell. As a result, it is not worth it to migrate or handover the service to another cell as the user moves. This is due to the delay constrains as well as the low mobility of the user by which increasing the transmission power of the cell can be a solution. In addition, the density of

the deployed server or cell can cause a lot of handover or sometimes outage of the cells, and this is due to the interference between the cell.

The work in [76] and [77] depart from the mentioned problems in the previous paragraph. In [76], a power control algorithm, Cloud-aware Power Control (CaPC), was proposed to avoid handover and outage situation when the channel quality between the mobile user and the connected cell changes. This algorithm is to guarantee that the mobile user is able to receive results of data computed by the Cloud on time by increasing the coverage of the connected cell. The CaPC is composed of coarse and fine setting. The purpose of coarse setting is to find the optimal default transmission power. The fine setting is based on short term adaptation of transmission power when the mobile user can not receive request from the Cloud due to the low Signal plus interference to noise ratio.

Two performance metrics were used to evaluate CaPC: number of undelivered requests and amount of data not served by the SCCs. When compared to more traditional power control systems, simulation results show that the amount of undelivered requests back to the mobile node can be greatly reduced. This work was extended in [77] to adaptively set the time instant when the CaPC is triggered. As a result, this will reduce the inference between cells and additional delay of delay sensitive tasks. According to the simulation evaluation, the number of applications that were successfully delivered increased by 98%.

2.4.7.2 High Mobility

In the high mobility scenario, migration is needed if the user moves away far from the connected MEC server. The studies in [78, 79, 80, 81, 82, 83, 84, 85, 86, 87] considered the problem of migration between the MEC servers with various objectives. In a situation where a huge amount of data must be migrated, the migration become inefficient, therefore path selection was considered in the studies [88, 89]. As the focus of this work is to select an Edge server for offloading, it is assumed that there exists a mobility management method to keep the mobile node connected to the selected server in case the mobile node moves out of the range of that server. The mobility scenario will be analysed further in Chapter 5.

2.5 Optimal Stopping Theory (OST)

In general, OST involves making a decision as to whether a given action should be taken based on a random variable that is sequentially observed, with the aim being to increase the expected payoff or to reduce the expected costs. The optimal stopping problem applies if the decisions are taken in stages and the result of each decision is not fully predictable, but can be estimated in certain ways prior to the next decision [90]. A key feature of such a situation

is that decisions can not be evaluated alone, but must be weighed against the need for low current costs and the undesirable of high future costs. The key difficulty in any optimal stopping problem is not which options to select, but how many options to explore [91]. In other words, in any optimal stopping problem, we try to define the optimal rule by which the decision-maker can take a decision to minimise or maximise an ultimate objective.

In principle, stopping problems are defined by two main objects: (1) a sequence of random variables, $X1, X2, ...,$ whose joint distribution is assumed to be known, and (2) a sequence of real-valued reward (or cost) functions [22]. Usually, therefore, the objective is the minimisation (or maximisation) of the defined function or the observed random variable itself. Stopping problems can be categorised based on the knowledge of the number of observations into finite horizon problem and infinite horizon problem.

The following subsections provide a preliminary introduction to the original models used in this thesis, their fundamental solutions and the motivation of using such models. Expanded explanation of the models' principles of each OST-based method adopted in this thesis and their applications to the considered problem will be presented in the following chapters.

2.5.1 Best-Choice Problem

In the Best-Choice Problem, the decision-maker is observing a sequence of objects which can be ranked from best to worst. The aim of the decision maker is to select a stopping rule that maximises the chance for the best object to be picked [22]. In a simple form, the solution of such a problem is to reject a proportion 37% of the objects and select the best object seen so far [22]. The motivation of adopting such a model in our case is that the model does not require a lot of information being available to the decision-maker in order for the decision to be made. In other words, the decision maker knows nothing about the observations other than how they compare to one another [91]. As a result, there are no requirements to store any information about the observations. Furthermore, the probability of choosing the best (e.g., the minimum processing time) is always at least 0.36% which is the highest rate of success one can achieve in such a setting, i.e., when no previous information about the observed random variable is available [22].

2.5.2 Odds Algorithm

The Odds algorithm deals with a sequence of independent events. The decision maker observes these events and categorises them as "success" or "failure" [92]. Similar to the case of the Best Choice problem, the decision-maker observes these events one by one and wants to decide which event should be taken. The decision maker cannot return to the rejected event.

The objective of the Odds algorithm is to find stopping rules that maximise the probability of picking the last success. The Odds for each observation refers to the probability of the success of the event divided by the probability of the failure of that event. Generally, the solution of this problem is to sum the Odds from the last the observation until the value 1. From that index, the decision-maker should start looking for a success. The Odds algorithm looks at applications where the last observation tends to have considerably more interests to the decision maker. The motivation for using the Odds is that, in our case, delaying the selection gives the decision-maker a better chance of improving the decision by exploring more options. In addition, based on the probability distribution of the random variable, it becomes easy to estimate the Odds of each observation. Further, the algorithm's performance still unaffected when integrating some factor (e.g., quality indicator) [92] as we shall see later in Chapter 3. The performance of the Odds algorithm is the same lower bound that is achieved by the Best-Choice model, i.e., there is at least around $1/e$ probability of choosing the last best observation [93].

2.5.3 Selling Asset Problem

In this problem, an individual, for example, has an asset for which he is offered an amount of money from period to another. The objective of this problem is to find a stopping rule that maximises the revenue of the person [90]. The assumption in this problem is that the offers are random, independent and identically distributed with known distribution. Also, the decision has to be made within pre-defined number of observations, i.e, finite horizon problem. An extension of this problem is where there is a cost with each observation and the stopping problem is infinite horizon, i.e., the decision-maker is only required to provide a cost for each observation. We still have the same assumption that the random variable to be optimised is independent and identically distributed with known distribution.

In principle, in these two problems, the solution is obtained by the method of the dynamic programming applying the principle of optimality and the optimality equation [22]. The dynamic programming, proposed by Richard Bellman back in 1952, divides a problem into sub-problems, and then combines the solutions for the sub-problems to get the overall solution [94]. The principle of optimality states that if the future optimal decisions are known, then one can obtain the optimal solution for the current stage, and compare it with the future decisions. For example, in the finite horizon stopping problem, the problem can be solved by the method of backward induction. Since we must stop at at the last stage, say n, we first find the optimal rule at stage $n-1$. Then, knowing the optimal rule at stage $n-1$, we find the optimal rule at stage $n-2$, and so on back to the initial stage (stage 0) [22]. In the infinite horizon stopping problem, the optimal rules can obtained by comparing the current value with the estimated value in the next step. The motivation of using such an approach,

i.e., selling asset problem, is the assumption that the decision maker will be able to obtain more statistical information about the observed random variable, resulting in significantly improved performance. In other words, having such information will make a significant difference in terms of achieving the desired objectives. Also, in this case, beside the ability to achieve the main objective, one can also take into account a delay factor for the decision or the cost of observations.

To sum up, in this thesis, the models presented above have been utilised and applied to the context of computation offloading in MEC environment. The core objective of this work is to make mobile nodes independent and to take offloading decisions without having to rely on a central controller or additional information about potential offloading sites. In general, the OST makes time-optimised decisions statistically in cases where no information or incomplete information is available, which makes it appropriate for independent and standalone decision-making with regard to selecting the best time or server for offloading and allowing easier implementation in mobile nodes. In particular, the OST models are simple to implement and only requires a linear search over the observations with time complexity of $\mathcal{O}(n)$ in the worst case scenario, i.e., when the decision maker picks the last option.

2.6 Summary

This Chapter places the thesis in the context of existing efforts on the concept of computation offloading. As described in Section 2.2, the Cloud may add additional costs to mobile node applications, and the trend now is thus towards the adoption of Edge computing. The latest proposals within the Edge computing paradigm include the concept of MEC computing, where Edge servers are deployed and integrated within mobile networks in order to provide a variety of services to mobile nodes while improving the general QoS. While Cloud servers can provide infinite and powerful centralised resources for mobile nodes, enhancing their processing power and battery lives, having Edge servers in place for task offloading makes computation offloading more practically beneficial than the use of traditional distant Cloud methods.

Accelerated by recent advances in terms of wireless communications and new mobile applications, the MEC paradigm is expected to host and implement multiple different types of applications. Examples of such applications include virtual and augmented reality and data analytic tasks focused on the large quantities of data generated by users in such complex environments. It is also expected that multiple deployment options will be used for MEC servers, such as base stations, small cells, and RSUs.

As described in Section 2.4, the point of offloading is to enhance mobile nodes' capabilities. Extensive efforts to deal with the problems related to the task offloading have been made, and examples of early frameworks include MAUI, ThinkAir and CloneCloud. These frameworks also coincided with the introduction of the Cloudlet, supporting the idea of task offloading to nearby resources.

Due to its dynamic nature, the MEC environment can be very challenging; in particular, several different types of decision-making are required during the offloading process. Therefore, as described in Section 2.4.5, different advanced optimisation methods have thus been utilised to deal with this, including, but not limited to, reinforcement learning. Some studies have considered the problem of selecting a server for offloading, while other recent studies have dealt with decision-making as a centralised method combined with decision-making dealing with other problems. This centralised method is generally considered to be at a disadvantage, however, not only due to it making the mobile node more dependent, but also because it introduces more load to the mobile network.

OST-based decision-making, which includes various different methods, is considered to offer lightweight algorithms that lead to optimal or near-optimal solutions for many decision-making problems. While some OST-based models require additional information to be available to the decision maker, this work argues that it is not difficult for a mobile node to obtain such information either by itself or with the help of the MEC service provider.

Table 2.5 identifies the main context of the thesis from different perspectives.

Aspect	Position
Edge computing concept	MEC
Edge server deployment	RSU and small cell deplyment
Offloading decision	Server (time) offloading selection
Offloading Metric	Processing time at the Edge server, load
Solution	Optimal Stopping Theory

Table 2.5: Scope of this thesis.

Based on the review presented in this Chapter, the main direction for this thesis is thus identified as follows:

- Recent offloading decision-making studies with respect to server or time selection have not been focused on making this decision-making autonomous in terms of being made by the mobile node itself in an independent manner. Therefore, utilising the characteristics of the OST-based models mainly the optimality and the dependability, this work presents a spectrum of the OST-based decision-making algorithms and how they can be applied to the task offloading decision as well as their performance evaluations.

In the next Chapter, the thesis presents the main design and the considerations for a lightweight and an optimal OST-based task offloading framework.

Chapter 3

Maximising the Probability of Best-Server Offloading

3.1 Overview

As outlined in Chapter 2, the MEC servers are envisaged to be deployed in RSUs in order to enhance mobile nodes by utilising computation task offloading. The MEC servers are equipped with storage and computing units [95]. This configuration can be seen in vehicular network applications, as studied in [5] and [6], where smart vehicles perform different types of tasks. For instance, a mobile node can offload contextually collected data to perform data analytics tasks on one of the MEC servers. A mobile node may refer to a smart vehicle or smartphone in the vehicle used by the passenger. The mobile node can access the RSUs via V2I mode utilising DSRC communication.

Furthermore, as discussed in the previous Chapter, existing studies on the decision-making of selecting an offloading site are either dependent on other nodes, in need of complete information of network and servers' status or implemented on the server side as a result of the complexity to obtain the global optimisation. Also, there are some proposals for a decision-making engine to be implemented in the mobile nodes. However, such implementations might be an extra overhead for the mobile node during the offloading decision-making process. These proposals might conflict with the main goal of offloading where we try to enhance the mobile node's resources capabilities without incurring any additional cost.

Moreover, because the MEC servers operate at the Edge within the RAN with the help of the RSUs, their coverage areas may be limited by the radio coverage of the RSUs [5]. Thus, the mobile node only knows about the current MEC server, i.e., the server in the range of that node. Unlike the centralised architecture, this work concerns the case where there is no centralised controller or server to assist the mobile node to make the decision. The mobile

node is responsible for making the offloading decision autonomously and locally. Thus, the proposed model is to be built on the top of an offloading decision framework implemented in the mobile node from previous work, which provides the entity of network and Edge servers profilers as observed in [25, 75]. Such profilers are adopted to provide information about the current load and or the experienced processing time of MEC servers.

In the context of this architecture and in order to deal with the aforementioned limitations, this Chapter presents two lightweight sequential offloading decision-making algorithms, the objective of which is to maximise the probability of offloading to the server with the best processing time. Section 3.2 presents the system model and the assessed setting, whilst Section 3.3 proposes a general model for task offloading decision-making. This model is inspired and based on the Best Choice Problem (BCP). In addition, in Section 3.4, data-quality based offloading decision model is proposed utilising the theory of Odds within the context of the OST.

3.2 System Model

A setting where there exists a finite set of MEC servers deployed along mobile nodes' paths on the move as shown in Figure 3.1 is considered. The mobile node is moving in One Dimensional (1D) mobility model and observes the MEC server sequentially (for instance, in one way road) as considered in [5]. Let X_k be the random variable indicating the processing time of k-th observed MEC server. X_k can indicate different random variables, e.g., the current load of the MEC server which affects the processing time of the task, the transmission time coupled with the computational workload of a MEC server or the time it takes the MEC server to broadcast the results to other system, e.g., real time information for transportation system. For simplicity, we call it processing time throughout the thesis unless otherwise specified.

Once a task[1] is locally generated and needs to be offloaded, then, at each time, within the number of observed MEC servers n, the mobile node checks the value of X_k for each MEC server k it passes by using network and server profilers. The mobile node needs to decide whether to offload to the current k-th server or continue observing another server. To keep the continuity of task processing, we assume that there is a mobility management entity in the server [13] which implements a mobility management algorithm, such as path selection, power control algorithms [13, 89] or predictive model as in [5]. For example, if the task involves getting some results from the MEC servers and the mobile node is out of the range of that MEC server, then the selected MEC server should be transmitting the results through

[1]Unless stated otherwise, the terms data offloading and task offloading are used interchangeably.

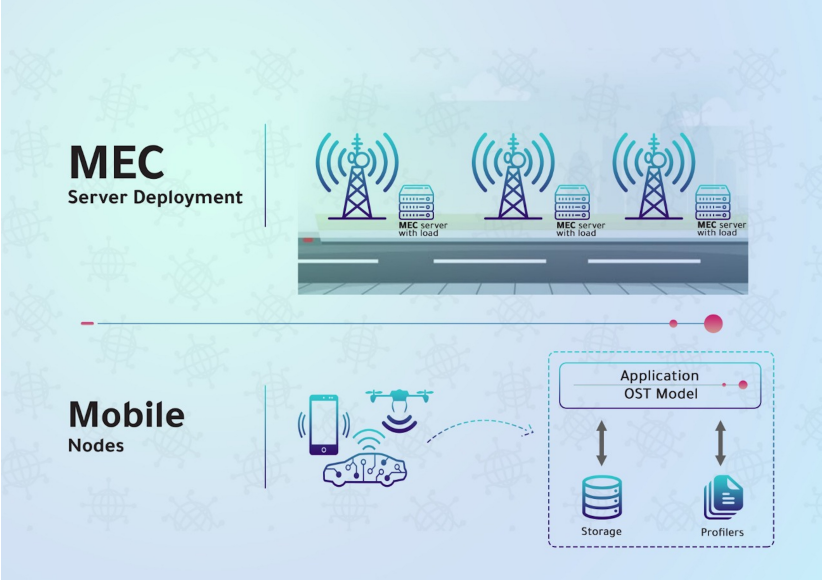


Figure 3.1: MEC servers and mobile node settings.

the next MEC server using high bandwidth wired connection [6]. In Table 3.1, we provide the key notations used in this Chapter.

Notation	Explanation
X	the random variable to be optimised
k	the index of the observed MEC server
n	the number of MEC servers or the number of observations
r_n	the optimal cutoff within the BPC model is taken by $r_n - 1$, r_n can also refer to the Odds in the observation n
$P_n^*(r_n)$	the maximum probability of ending up with the best server when applying the BCP policy
r_k	the Odds for observation k
P_k	the probability of having X_k less than or equal to a threshold within the Odds model
f	data quality indicators (timeliness)
s	the stopping threshold in the Odds model from which we start check a MEC server
$P_s^*(r_s)$	the maximum probability of ending up with a server meeting the required threshold when applying the Odds policy
R_s	the sum of the Odds from n until we reach or exceed the value 1
Q_s	the product of the complementary probability $(1 - P_k f_k)$ from n until s
θ	the threshold required within the Odds model
δ	the probability of having less than or equal to θ

Table 3.1: Key notations used in Chapter 3.

3.3 Best-Choice Problem Based Task Offloading

The first model deals with the case that the number $n > 0$ of the available MEC servers, which are candidates for task offloading, is known to a mobile node. Now, the goal is to optimise the decision on when to offload the computational task to an available server. Formally, our objective is to maximise the probability of offloading to the optimal server. The mobile node is on-line observing a sequence of candidate MEC servers, which are locally ranked in the node from the best to the worst w.r.t. a performance criterion, i.e., the current processing time X. At each observation, the mobile node should decide whether to choose

the current available candidate MEC server or not. In the latter case, as the mobile node is moving in 1D mobility model, the node cannot recall its decision, i.e., if a candidate MEC server is rejected for selection, it cannot be recalled.

The challenge now is that the mobile node desires to define an offloading policy (rule) which maximises the chance of choosing the best MEC server w.r.t. the ranking seen so far. Every MEC server is relatively ranked based on the previous observed MEC servers and can only be checked sequentially and in a random order. Once a MEC server is relatively ranked and rejected, this choice cannot be re-called. The mobile node should maximise the probability to select the candidate among the n candidates, which is globally ranked best. This is cast as a Best-Choice Problem (BCP). In the adopted BCP, the goal is to find the offloading rule that maximises the probability P_n^* of selecting the best of all n servers and the corresponding probability of that success.

3.3.1 Problem Formulation

Let us call the k-th server *candidate*, if it is relatively best in terms of X_k, $k = 1, \ldots, n$. We then define a positive integer $r_n \in \{1, \ldots, n\}$, defined as:

$$r_n \;=\; \min\{r \geq 1 : \frac{1}{r} + \frac{1}{r+1} + \cdots + \frac{1}{n-1} \leq 1\}, \tag{3.1}$$

for $n \geq 2$. Based on the BCP, the optimal policy is to reject the first $r_n - 1$ servers and then select the first candidate, if any, to offload the tasks. For reasons of completeness, we provide **Theorem 1** and **Theorem 2** [2], where the optimality of the BCP model is based on.

Theorem 1. *The maximum probability of selecting the best candidate in the BCP in (3.1) is given by:*

$$P_n^*(r_n) \;=\; \frac{r_n - 1}{n} \sum_{k=r_n}^{n} \frac{1}{k-1} \tag{3.2}$$

Proof. The probability of offloading to the best server is $\frac{1}{n}$ for $r = 1$, and, for $r > 1$,

$$\begin{aligned} P(r) &= \sum_{k=r_n}^{n} P(\text{server } k \text{ is selected and server } k \text{ is the best}) \\ &= \sum_{k=r_n}^{n} \left(\frac{1}{n}\right)\left(\frac{r_n - 1}{k-1}\right) = \frac{r_n - 1}{n} \sum_{k=r_n}^{n} \frac{1}{k-1} \end{aligned}$$

□

[2] For more information about Theorem 1 and Theorem 2 see [22] and [96].

Theorem 2. *For a small value of n, the optimal r_n can be computed using (3.1). When $n \to \infty$, we obtain the well-known Secretary Problem where the optimal number of observations one should reject is $\frac{1}{e}$ and the probability of success tends to be $\frac{1}{e}$.*

Proof. Let $n \to \infty$, $\frac{1}{k-1}$ as $\frac{1}{x}$ and x as $\frac{r_n - 1}{n}$, the probability function $P(r)$ can be approximated by the integral: $P(r) = x \int_{r-1}^{n} \frac{1}{x} dx = x \ln \frac{1}{x} = -x \ln(x)$. By finding the derivative of $P(x)$ with respect to x and set it to 0, we have $-\ln(x) - 1 = 0$, and $\ln(x) = -1$, and by taking the exponent for both sides, we have $x = \frac{1}{e}$. □

3.3.2 Optimal Task Offloading Rule

BCP-based Optimal Task Offloading Rule: The mobile node observes the first random n/e MEC servers and ranks them immediately w.r.t. their processing time provided by each of them upon request. Then, the mobile node offloads their task/data to the first t-th MEC server with $k > \lceil n/e \rceil$ which is ranked as the relatively best server compared to the previously observed servers.

Based on this optimal offloading policy, the node is guaranteed to maximise the probability of offloading the task/data to the best MEC server. If no offloading decision is made after observing the n MEC servers, the node offloads the task/data to the n-th MEC server, since no recall is allowed.

3.3.3 Analysis

Let us consider an example with a finite small n, e.g., $n = 3$. That is, there are 3 MEC servers in the mobile node's path. These servers have different processing times. We refer to the MEC server with the minimum processing time ranked with the number 1, and the server with the highest processing time ranked with the number 3. The MEC servers might come in different order. Thus, we have 6 permutations $(3!)$. One policy is to reject the first server and take it as baseline and then accept the first relatively best server after that. If we follow such policy, 50% of time, we select the best one. In other words, there is 50% chance of offloading to best. If we increase the number of MEC servers by only 1 and follow the same policy, the chance of offloading to the best becomes close to 45%. As the number of MEC servers gets larger, the chance of offloading to the best gets smaller. Therefore, such policy does not work well with larger number of MEC servers and does not give the maximum probability which decreases with the number of MEC servers involved. Moreover, if the mobile node

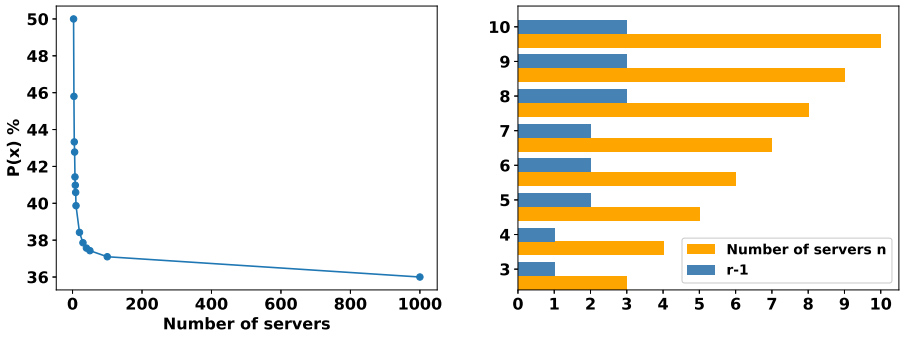


Figure 3.2: The probability of offloading to the best (left) and the value of $r-1$ (right) for different numbers of MEC servers n [91].

is offloading randomly to one of the encountered servers for e.g., $n = 10$ MEC servers, we have only 10% chance of offloading to the best server.

In Figure 3.2, we show the (desired maximum) probability of offloading to the best (left) and the value of $r-1$ (right) for different numbers of MEC servers n. We observe that as the number of MEC servers grows, we end up with the 36% chance of offloading to the best [91]. In reality, we expect the mobile node to have a number of MEC servers less than 10, thus, following the BCP's rule, we have a probability of offloading to the best $\geq 39\%$. Moreover, the BCP model only requires the number of observations n the mobile node is willing to observes. The number of observations can be defined and fed to the BCP model by the mobile node itself. Therefore, the decision-making algorithm in this model is very independent and does not require relatively a lot of information.

Nevertheless, in the MEC environment, we expect that there will be other information, available to the mobile node, in addition to the number of MEC severs n. Such information can be utilised to adopt and apply an advanced model within the context of OST. Moreover, it is possible to consider other requirements with a better performance in the decision of task offloading. Therefore, in the following section, motivated by the BCP model and aiming to achieve and optimise the same objective, i.e., maximising the probability of offloading to the MEC server with the minimum processing time, we provide an advanced decision-making algorithm where the mobile node can provide more information in order to have better results.

3.4 Quality-aware Contextual Data Offloading

In this section, we study the case that arises when a mobile node desires to offload contextual data to a MEC server and performs data analytics task while on the move. The data analytics

task can be data correlation analysis, inferential and predictive analytic [97], statistical learning models building, model selection [98, 99] or data for HD maps as in [95]. The data can be gathered via different applications such as a MCS or vehicular crowd-sensing [55, 100]. For example, in vehicular crowd-sensing applications, vehicles sense data from surrounding environment, process them and send the processed results within a specific deadline to a centralised application manager for further processing [100]. In this use case, beside the main objective (maximising the probability of offloading to the best server), the mobile node wants to offload contextual data to perform an analytic task *before* the data turns obsolete (stale). To deal with this quality of analytics problem, we elaborate on the the Odds algorithm within the context of the OST enhanced with a data quality indicator in the task offloading decision.

The Odds algorithm is an OST algorithm for computing optimal stopping rules in order to maximise the probability of stopping at the last observation which satisfies a specific criterion [101]. We call an observation that satisfies the defined criterion a *success*. Specifically, the traditional Odds-algorithm applies to a class of problems called last-success-problems. The objective is to maximise the probability of identifying in a sequence of sequentially observed independent events the last event satisfying a specific criterion.

3.4.1 Problem Formulation

Let $f : \mathbb{T} = \{1, 2, \ldots\} \rightarrow [0, 1]$ represent how *stale* the data is, which is a non-increasing function adopted from [102]:

- f is non-increasing in $\mathbb{T}$,
- $f_0 = 1$, where $k = 0$ is the start time before collecting the first data,
- $f_n = 0$, for $k \geq n$.

A linear timeliness function f is as follows:

$$f_k = \begin{cases} 1 - \frac{k}{n+1}, & 1 \leq k < n. \\ 0, & k \geq n. \end{cases} \tag{3.3}$$

To get more insight on the Odds algorithm, let us consider a mobile node that is sensing data while on the move. The mobile node is trying to offload the data to a MEC server that has a processing time less than or equal to a desired threshold θ defined by the application launched on the mobile node. The Odds of the observed server k denoted by r_k is defined as the ratio of the probability P_k of having the MEC server with a processing time X less than

or equal to θ divided by its complementary probability $1 - P_k$ [92]. Specifically, the Odds at time k is defined as follows:

$$r_k = \frac{P_k}{1 - P_k}, P_k < 1 \tag{3.4}$$

In each observation k, we take into account the Odds $r_1, \ldots, r_k$ as well as the timeliness $f_1, \ldots, f_k$ of the collected data by evaluating the function in (3.3). Hence, we obtain that:

$$r_k = \frac{P_k f_k}{1 - P_k f_k}, \tag{3.5}$$

where the Odds r_k depends now on how stale the data are at time instance k. Let $P_k = P(X_k \leq \theta)$ denoted by δ, then we have:

$$r_k = \frac{\delta f_k}{1 - \delta f_k} \tag{3.6}$$

(3.3) can be reformed as:

$$f_k = 1 - \frac{k}{n+1} = \frac{n+1-k}{n+1} \tag{3.7}$$

Then, we can substitute (3.7) for f_k in (3.6):

$$r_k = \frac{\delta \frac{n+1-k}{n+1}}{1 - \delta \frac{n+1-k}{n+1}} \tag{3.8}$$

(3.8) can be simplified as:

$$r_k = \frac{\delta(n+1) - \delta k}{(1-\delta)(n+1) + \delta k}. \tag{3.9}$$

Note that the Odds r_k changing with the time (k) as a non-linear function of the observation k reflecting the constraints of the data timeliness while engaging the application specific threshold $\delta = P(X_k \leq \theta)$ for assessing the appropriateness of the k-th MEC server. Figure 3.3 shows the evolution of the Odds r_k against observation k for different δ values in $\{0.3, 0.5, 0.8\}$ with $n = 10$. The Odds values decrease as we approach the end of (candidate) MEC observations, while a high application threshold $\delta = P(X_k \leq \theta)$ increases the Odds at the beginning of the selection process (being optimistic due to a relatively high δ). However, as $k \to n$, the Odds shrink to a very low value to *enforce* the decision of offloading to be taken, thus, avoiding offloading stale data (at $k = n$ with $f_n = 0$).

Let us now elaborate on the modified Odds algorithm that takes into consideration the data

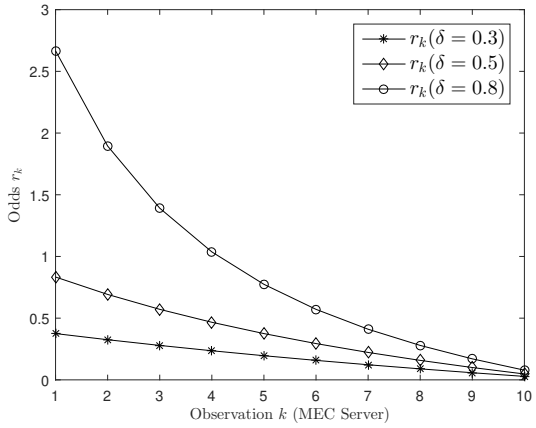


Figure 3.3: The Odds r_k against observation k for different δ values; $n = 10$.

timeliness indicator f_k in the optimal task offloading decision. In our context, we aim at maximising the probability of offloading to the *last* MEC server with $P_k = P(X_k \leq \theta)$ for a given θ threshold such that P_k is higher than all preceding probabilities $P_l, l = 1, \ldots, k-1$ seen so far. And, this decision must be taken at the time of observation. Hence, the very last MEC server with the above-mentioned criterion is the highest *bid*. Maximising the probability of offloading on the last k-th MEC server with $P(X_k \leq \theta)$ therefore means maximising the probability of offloading to the best MEC server w.r.t. θ.

In the context of the computation offloading, the Odds might be unknown but it can be estimated based on the probability distribution function of the random variable to be optimised. According to Odds theorem [92], in the case where the Odds is unknown, the problem still makes perfect sense as long as r_k can be estimated. For example, when P_k is of the form $P_k = P f_k$ where f_k is known. In the proposed quality-aware data offloading, we principally add the timeliness indicator f_k to the P_k, thus, now r_k represents the Odds of the k-th event turning out to be candidate for data offloading.

The Odds-algorithm sums up the Odds in reverse order:

$$r_n + r_{n-1} + r_{n-2} + \cdots ,$$

until this sum reaches or exceeds the value 1 for the *first* time. Let us denote that this happens at observation s, i.e., the corresponding sum R_s exceeds 1 with

$$R_s \quad = \quad r_n + r_{n-1} + r_{n-2} + \cdots + r_s. \tag{3.10}$$

If R_s does not reach 1, then we set $s = 1$. Also, at the same time we compute the product:

$$Q_s \quad = \quad \prod_{k=s}^{n} (1 - P_k f_k). \tag{3.11}$$

As we will see in the following subsection, and based on the Odds-algorithm in [92], the sum R_s and the product Q_s are required to calculate the optimal win probability, i.e., the probability of offloading to the optimal server. Based on the R_s and Q_s, we then apply the Odds algorithm to determine the optimal strategy for offloading to the best MEC server.

3.4.2 Optimal Task Offloading Rule

Quality-aware Odds Optimal Task Offloading Rule: The mobile node observes the MEC servers one after the other and decides to stop on the first MEC server from time s onwards (if any), where s is the stopping threshold such that $R_s \geq 1$.

This strategy is optimal, that is, it maximises the probability of stopping on the last best MEC server. And, this is happening with the maximum probability which equals to:

$$P_s^*(r_s) \quad = \quad Q_s R_s : R_s \geq 1. \tag{3.12}$$

Note that $P_s^*(r_s)$ is always at least 0.368 and this lower bound is best possible, which is achieved by the BCP policy with a very large number of MEC servers n [93]. It is also worth mentioning that the Odds-algorithm computes the optimal strategy and the optimal probability $P_s^*(r_s)$ at the same time. Also, the number of operations of the Odds-algorithm is sublinear in n.

In practice, the mobile node should reject the observations (MEC servers) from $k = 1$ until s and from the observation s, the mobile node starts checking each observation (candidate MEC server). If it is a success, i.e., $X_k \leq \theta$ for $k > s$, then, the mobile node should offload the data to the k-th MEC server, otherwise it continues observing until f_n, i.e., where the data must be offloaded since $f_n = 0$.

3.4.3 Analysis

As an example, assume that MEC processing time X follows normal distribution with mean 50 ms, a standard deviation of 10 ms ($X \approx \mathcal{N}(50, 10)$) for a specific data size and analytics tasks and the data on the mobile node must be offloaded within the next $n = 10$ observations. The timeliness of the data can be specified by the task application (it can be the number of time intervals or it can be the number of MEC servers the mobile node should observe before $f_n = 0$). If we assume that the mobile node will have $n = 10$ observations and the mobile node is looking for a MEC server with processing time less than $\theta = 50$, then, based on the Odds algorithm enhanced with f_k timeliness indicators, the strategy suggests to start looking for a MEC server to offload from $k = 5$ and onward. By doing this, there is $\approx 42\%$ chance

of offloading to the (last) best MEC server , which is the maximum that can be achieved. The probability here refers to the situation where we end up with a MEC server with processing time less than 50, thus, satisfying our criteria.

In real world scenarios and in the long run of a mobile node application, the MEC servers' provider can provide the probability distributions of the random variable X_k of the MEC servers based on the locations of the mobile node. Alternatively, the mobile node itself can use the historical data of the task offloading to learn the probability distribution. Once we can estimate the probability distribution of the processing time, the mobile node can estimate, based on the model above, where it should start checking the performance criterion in order to maximise the probability of offloading to a MEC server that meets the defined condition.

3.5 Summary

In this Chapter, two optimal task offloading rules are derived based on the principle of the OST: the BCP-based and the Odds-based models. To review the proposed models, they are being visually described in the Figures 3.4 and 3.5. As shown in Figure 3.4, the BCP model takes the number of observations n as an input and outputs the numbers of servers that should be rejected before considering a MEC server for offloading. The mobile node should offload if the processing time X_k is the best seen so far, otherwise, the mobile node should continue observing until the server n. By that time, the mobile node must offload to server n.

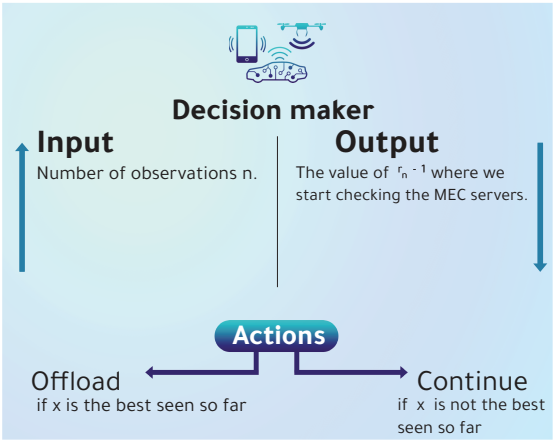


Figure 3.4: BCP.

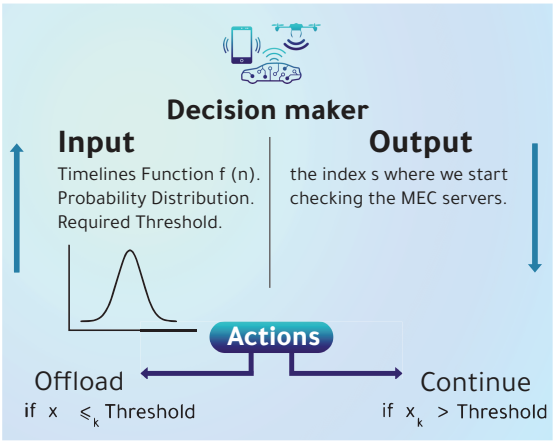


Figure 3.5: Quality-aware Odds Model.

The proposed data quality-aware Odds model, shown in Figure 3.5, takes the probability distribution function, a timeliness function (indicator) f_k and a defined threshold as inputs and outputs the numbers of servers $s < n$ that should be rejected before considering a MEC server for offloading. The mobile node then starts evaluating the condition based on the required threshold θ. If the condition is true, then, the mobile node should offload, otherwise,

the mobile node should continue observing until $f_n = 0$. By that time, the mobile node must offload before the data turns obsolete.

Overall, the proposed models' principle presented in this Chapter will enable mobile nodes to make a decision on the selection of the MEC server to be used for offloading exploiting the mobility and the deadline for the task (for instance, timeliness of the data) locally and independently, while minimising the need for a centralised server to help in the offloading decision process. In the following Chapter, the objective of minimising the processing time is considered.

Chapter 4

Minimising the Processing Time

4.1 Overview

Offloading to the optimal server, i.e., the server with the minimum expected processing time, is not an easy task. Following an offloading policy to minimise the expected processing time might be more beneficial with a better performance. Now, the challenge is to find an optimal policy with the objective being the minimisation of the expected processing time. This is different from the previous policy presented in Chapter 3 (maximising the probability of offloading to the server with the minimum processing time), since now we care for minimisation of the sequentially observed processing time values.

Similar to previous Chapter, the case where the mobile node is moving in 1D mobility model and passing by a set of MEC servers is studied. The mobile node connects to the RSUs via V2I communication mode using DSRC. The mobile node is to make the selection of the MEC server locally and independently. Network and Edge servers profilers are utilised to check the processing time of the MEC servers.

This Chapter presents two sequential offloading decison-making algorithms with the objective of minimising the processing time. First, a Delay-tolerant Task Offloading (DTO) is proposed in Section 4.3. This model considers the case where the number of observation n is known in advanced. In Section 4.4, however, a Cost-Based Task (COT) model is proposed where the assumption of having the number of observation available to the mobile node is dropped. Instead, the mobile node provides a cost per observation. For each model, the problem is formulated and the optimal offloading rule is obtained.

Notation	Explanation
X	the random variable to be optimised
k	the index of the observed MEC server
n	the number of MEC servers or the number of observations
z_k	system state at time k
r	delay factor for the decision maker in the DTO model
a_k	the threshold value at observation k in the DTO model
a_n	the threshold value at observation n in the DTO model
c	the cost per observation in the COT model
V^*	the optimal threshold in the COT model

Table 4.1: Key notations used in Chapter 4.

4.2 System Model

Similar to previous Chapter, a MEC system as shown in Figure 4.1, where a mobile node can offload data to perform a computing task on a specific MEC server, is considered[1]. The offloaded tasks can be perception tasks, computing tasks over offloaded data e.g., image recognition, image processing, data correlation analysis, inferential and predictive analytics [97], statistical learning models building and/or models selection [98], [99]. The mobile node can be a smart vehicle as proposed in [5] or smart phone used by the passenger of the vehicles. For each MEC server, at each time instance, there is a temporal load associated with it which affect the processing time X. X_k can indicate different random variables, e.g., the transmission time coupled with the computational workload of a server or the time it takes the MEC server to broadcast the results to other system, e.g., real time information for transportation system. For each observation k, there is a cost c the mobile node pays. For example, this can be a time cost, connection cost or delaying cost. A summary of notations used in this Chapter is shown in Table 4.1.

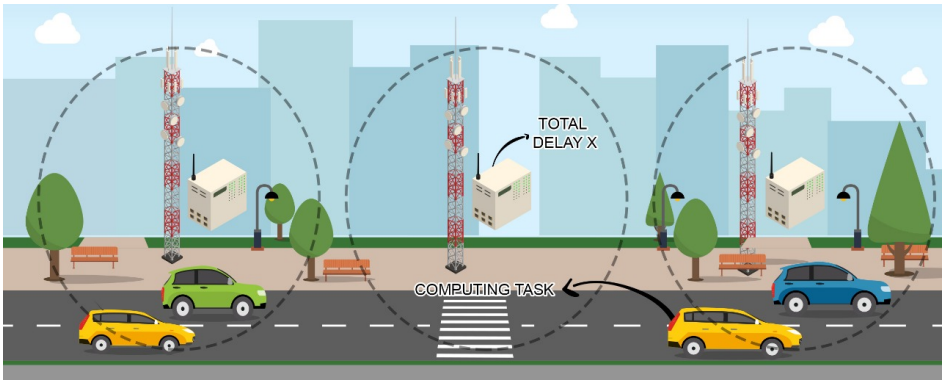


Figure 4.1: MEC in vehicular network.

[1]The models presented in Chapters 3 and 4 can also be generalised to other use cases such as the small cells deployment where multiple MEC servers are available to the mobile node.

4.3 Delay-Tolerant Task Offloading

This model also deals with the case that the number $n > 0$ of the available MEC servers, which are candidates for task offloading, is known to the decision maker, i.e., the mobile node. Similar to the previous Chapter, the goal is to optimise the decision on when (or which server) to offload the tasks to an available server. However, instead of focusing on the objective of maximising the probability of offloading to the best, in this model, the objective is to minimise the processing time X. The problem is modelled as a finite horizon OST problem.

In general, in the proposed models, including the models proposed in Chapter 3, the case where the expected processing time at the MEC server is lower than the expected processing time when executing the tasks locally on the mobile device is considered. In other words, the mobile node wishes to offload tasks and data for processing to a MEC server as it does not have the computational capabilities to do so and (or) sufficient energy for such tasks. Initially, the mobile node is observing the processing time for each server and it can either offload or postpone the offloading in light of having a minimised processing time.

4.3.1 Problem Formulation

The mobile node should find the best time instance k^* (server) such that the expected processing time $\mathbb{E}[X_k]$ is minimised, i.e., the optimal stopping time k^* such that the following infimum is attained:

$$\operatorname{ess\,inf}_k \mathbb{E}[X_k] \tag{4.1}$$

In general, the essential infimum refers to the greatest lower bound of a function. More specifically, the definition of essential infimum is as follows [22]:

Let X_k, for $k \in n$, be a collection of random variables. We say that a random variable G is an essential infimum of $(X_k)_{k\in n}$ and write $G = \operatorname{ess\,inf}_{k\in n} X_k$, if

- $P(G \leq X_k) = 1$ for all $k \in n$, and
- if G' is any other random variable such that $P(G' \leq X_k) = 1$ for all $k \in n$, then $P(G' \leq G) = 1$.

Hence, the expected minimum processing time is $\mathbb{E}[X_k]$ with $k = \inf\{k : X_k < \mathbb{E}[X_{k+1}]\}$. The problem here is a finite horizon stopping problem similar to the problem presented in subsection 2.5.3.

A discrete-time dynamic system, which expresses the evolution of a scalar variable, hereinafter referred to as the system's "state" z_k, under the influence of decisions made at discrete instances of time associated with the kth observation is defined. A state z_k summarises past information that is needed for future optimisation. By writing that the system is at state $z_k = X^*_{k-1}$ at $k \leq n$, we mean that the decision maker has not offloaded the data to a MEC server. By writing that the system is at state z_k = z_T, we mean that the decision maker has already offloaded the data to a MEC server, $k \leq n$, where z_T is defined as the terminating state. We take $z_1 = 0$ (a fictitious state). With these conventions (adopted from [90] and applied in [103]), the system equation (the mechanism by which the system is updated) has the form:

$$z_{k+1} = \begin{cases} z_T, & \text{if } z_k = z_T \text{ (stop).} \\ z_k, & \text{otherwise (continue).} \end{cases} \tag{4.2}$$

Let $J_k(z_k)$ be the optimal server to offload data to. The Bellman's equation for this system is then:

$$J_n(z_n) = z_n \tag{4.3}$$

for $k = n$, and

$$J_k(z_k) = \min \left[(1+r)^{n-k} z_k, \mathbb{E}[J_{k+1}(X^*_k)] \right] \tag{4.4}$$

for $k = 1, ..., n-1$.

Note that $\mathbb{E}[J_{k+1}(X^*_k)] = \mathbb{E}[J_{k+1}(z_{k+1})]$. The $r \in [0, 1]$ parameter is a *delay* factor, which prompts the decision maker to delay its optimal decision. In our model, a smaller *r* value denotes that the decision maker will skip a relatively small number of observations before proceeding with a offloading decision. The term $(1+r)^{n-k} z_k$ in 4.4 denotes the risk if the offloading happens at k and $\mathbb{E}[J_{k+1}(X^*_k)]$ denotes the expected risk if the decision maker continues the observation process. Hence, it is optimal to stop at stage k if

$$z_k \leq a_k = \frac{\mathbb{E}[J_{k+1}(X^*_k)]}{(1+r)^{n-k}} \tag{4.5}$$

else, it is optimal to continue.

4.3.2 Optimal Task Offloading Rule

The optimal stopping rule is determined by the scalar values $a_1, a_2, \ldots, a_n$ through which the mobile node decides either to offload or not. Specifically, the optimal offloading rule of the mobile node is:

Optimal Task Offloading Rule: stop the observation and offload the data at the k-th MEC server if $z_k \leq a_k$; otherwise continue the observation if $z_k > a_k$.

In our context, the optimal stopping rule states that the offloading decision (stopping) should happens right after receiving the k-th observation for which the expected processing time $X_k \leq a_k$.

The scalar variable a_k values are calculated as shown in Lemma 1.

Lemma 1: The scalar values $a_1, a_2, \ldots, a_n$ can be calculated once through the method of backward induction for $k = n$ to 1 from [2]

$$a_k = \frac{1}{1+r}\left(a_{k+1}(1 - F(a_{k+1})) + \int_0^{a_{k+1}} x dF(X)\right) \tag{4.6}$$

$$a_n = \frac{1}{1+r}\int_0^1 x dF(X) = \frac{1}{1+r}\mathbb{E}[X] \tag{4.7}$$

where $F(X) = P(X^* \leq X)$ is the cumulative distribution function of X^*.

Proof. Consider the function:

$$F_k(z_k) = \frac{J_k(z_k)}{(1+r)^{n-k}}, z_k \neq z_T$$

and then E$[F_{k+1}(X)]$ =E$[J_{k+1}(X)]/(1+r)^{n-k-1} = a_k/(1+r)^{-1}$ or $a_k = (1+r)^{-1}$E$[F_{k+1}(X)]$. Hence, we have $F_n(z_n) = J_n(z_n) = z_n$ and by placing E$[J_{k+1}(X)]$ with E$[F_{k+1}(X)](1 + r)^{n-k-1} in$ 4.4, for $k = 1, \ldots, n-1$, we get

$$\begin{aligned} F_k(z_k) &= \min\left[z_k, (1+r)^{-1}E[F_{k+1}(X)]\right] \\ &= \min(z_k, a_k) \end{aligned}$$

The a_k value is recursively calculated as follows:

$$\begin{aligned} a_k &= (1+r)^{-1}E[F_{k+1}(X)] \\ &= (1+r)^{-1}E[\min(X, a_{k+1})] \\ &= \frac{1}{1+r}\left(\int_0^{a_{k+1}} x dF(X) + \int_{a_{k+1}}^1 a_{k+1} dF(X)\right) \\ &= \frac{1}{1+r}\left(a_{k+1}(1 - F(a_{k+1})) + \int_0^{a_{k+1}} x dF(X)\right) \end{aligned}$$

[2]This solution is when X uniformly distributed in $[0, 1]$. Thus, the value of a_k in 4.5 is obtained based on the probability distribution function.

The scalar values $a_1, a_2, \ldots, a_n$ can be inductively obtained for $k = n$ down to 1, and the terminal condition is $a_n = \frac{1}{1+r}\mathbb{E}[X]$. □

4.3.3 Solution Principle

Before calculating the offloading rule, the probability distribution of the random variable (processing time X_k in our case) has to be known or estimated. For example, if the processing time X_k is uniformly distributed, the agent (mobile node) would first get the cumulative distribution using:

$$F(c \leq X \leq d) = \int_c^d f(x)dx = \frac{1}{b-a}dx = \frac{d-c}{b-a} \tag{4.8}$$

with $a \leq c < d \leq b$. After that, we calculate the expected processing time using:

$$\mathbb{E}(X) = \int_a^b f(x)dx = \int_a^b \frac{x}{b-a}dx = \frac{b-a}{2}, \tag{4.9}$$

were $a \leq X \leq b$.

When the random variable is normally distributed with known mean and standard deviation, the mobile node follows the same steps in the uniform distribution in order to get the a values. The cumulative distribution function of the normal distribution can be calculated using:

$$F(X) = \int_{-\infty}^{x} f(x)dx = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2} \tag{4.10}$$

When the random variable is exponentially distributed with known mean, the mobile node follows the same steps in the above distributions in order to get the a values. The cumulative distribution function of the exponential distribution can be calculated using:

$$F(X) = \int_0^x f(x)dx = \lambda e^{(-\lambda x)} \tag{4.11}$$

4.3.4 Analysis

For example, if the mobile node is having an idea that the X_k in a specific time interval is uniformly distributed between $a = 1$ and $b = 20$ seconds (or milliseconds) by studying the previous X_k of the same servers at similar time, it starts obtaining the scalar variable a_n and a_k by the backward induction method using equations (4.6) and (4.7).

The scalar decision values $\{a_k\}_{k=1}^n$ are illustrated in Figure 4.2. Now, it is optimal to offload at time k, i.e., on the k-th MEC, if the processing time $X_k \leq a_k$; otherwise, continue. In

other words, it is optimal to stop if the value of X_k is under the curve shown in Figure 4.2. By doing this, we are minimising the expected processing time. Figure 4.3 shows the a values when the random variable, i.e., X, is normally distributed and Figure 4.4 shows the a values when the random variable, i.e., X, follows exponential distribution.

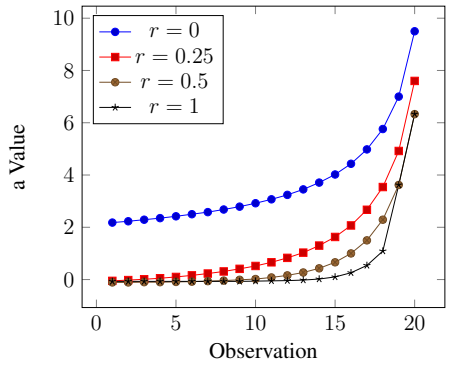


Figure 4.2: The values of the decision scalars $\{a_k\}_{k=1}^{n}$ for $n = 20$ observations based on a uniform distribution of X for different delay factors r.

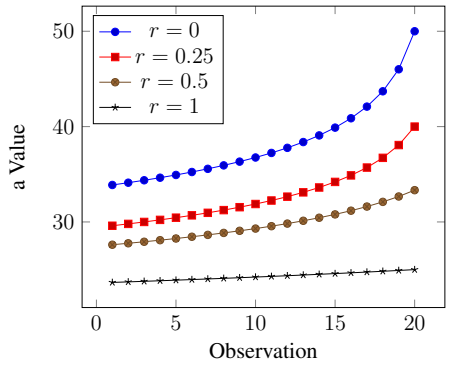


Figure 4.3: The values of the decision scalars $\{a_k\}_{k=1}^{n}$ for $n = 20$ observations based on a normal distribution of X for different delay factors r.

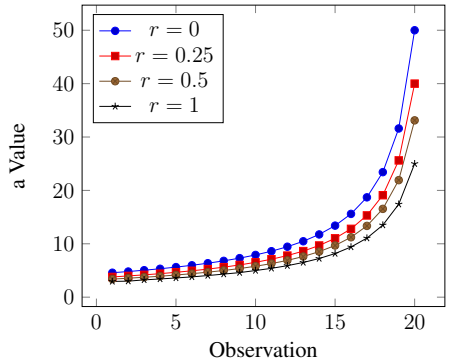


Figure 4.4: The values of the decision scalars $\{a_k\}_{k=1}^{n}$ for $n = 20$ observations based on an exponential distribution of X with mean of 50 for different delay factors r.

As an example, Figure. 4.5 shows the values of $\{a_k\}_{k=1}^{50}$ with the expected processing time X_k following normal distribution for $\mu = 50$ and $\sigma = 10$ when $n = 50$; it is optimal to offload when $k = 27$ as it is the first time the condition $x_{27} < a_{27}$ holds true. After we obtain the values of a_k, the value of x_k is checked. If $x_k \leq a_k$, the mobile node selects the MEC server k for data offloading, otherwise, it continues to the next available MEC server. At $k = n$, if the mobile node has not yet offloaded the data, it offloads the data to the first available MEC server (n-th MEC server).

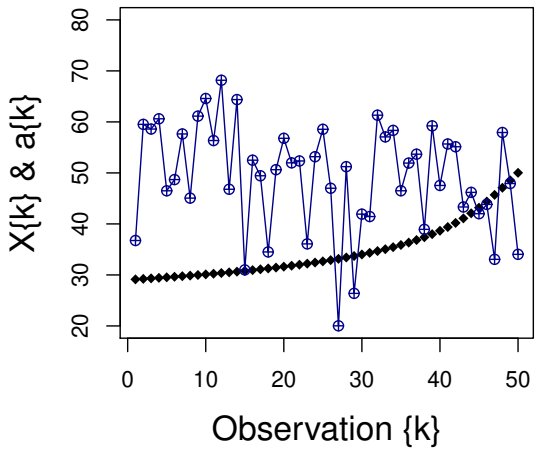


Figure 4.5: The decision values $\{a_k\}_{k=1}^{50}$ (black points) and simulated server delay or load X_k (blue points) vs. observations k for horizon $n = 50$; the optimal data offloading time when $k = 27, 29, 46, 47, 48$ and 50 where $X < a$.

4.4 Cost-Based Task Offloading

In section 4.3, we dealt with the processing time minimisation with the assumption that the number of the servers n is known to the mobile node. We now drop this assumption and contribute with an optimal policy for processing time minimisation where the number of servers (observations) is unavailable to the mobile node. Instead, the mobile node pays a cost c for each observation. Therefore, in this model, there will be a threshold for each cost. In general, the representation of the cost value depends on the application as we shall see in the next subsections. This problem is an infinite horizon optimal stopping problem as discussed in subsection 2.5.3.

4.4.1 Problem Formulation

Let X_k be the random variable denoting the processing time the node is observing for the k-th server at time k. We desire to find when to offload and which server that minimises the expected $\mathbb{E}[X]$. However, the node *pays* c cost units per observation when it has not yet offloaded the task. The cost can be application specific e.g., the rate the gathered data turn obsolete before being processed, a degree of urgency for task computation, the cost for requiring access to a server to ask for its current load.

We then define the cost function Y_k at time k including the defined cost up to k and the processing time X_k of the k-th server as:

$$Y_k \quad = \quad X_k + ck. \tag{4.12}$$

The target is to find the optimal offloading time $k^* = \arg\min_{k \geq 1} \mathbb{E}[Y_k]$ to decide to offload task to the k^*-th server such that up to k^* the expected cost $\mathbb{E}[Y_{k^*}]$ in (4.12) is minimised.

4.4.2 Optimal Task Offloading Rule

Optimal Task Offloading Rule: The node minimises the expected cost in (4.12) by offloading at the first k-th server such that:

$$k^* \quad = \quad \min\{k > 0 : X_k \leq V^*\} \tag{4.13}$$

where the V^* is the solution of:

$$\int_{V^*}^{\infty} (x - V^*) dF(x) = c. \tag{4.14}$$

where $F(x) = \int p(x)dx$ is the Cumulative Density Function (CDF) of the processing time X.

4.4.3 Solution Principle

The node offloads the task to the first server whose $X_k \leq V^*$, where the V^* value is the solution of (4.14). For instance, given a uniformly distributed load $X \in [0, 1]$, i.e., $F(x) = x$ and thus $dF(x) = dx$, we obtain for $V^* \in [0, 1]$:

$$\int_{V^*}^{1} (x - V^*) dF(x) = (1 - V^{*2})/2$$

while for $V^* < 0$:

$$\int_0^1 (x - V^*) dF(x) = 1/2 - V^*$$

Hence, based on the optimal task offloading rule in subsection 4.4.2,

$$V^* = 1 - (2c)^{(1/2)}$$

for $c \leq 1/2$, and:

$$V^* = -c + (1/2)$$

for $c > 1/2$.

After calculating V^* for a given cost c, the node starts off the load observation (or processing time) per server, one at a time: If it is less than V^*, the node offloads the task; otherwise, it continues checking for a better load before a pre-defined deadline. When the node has not yet offloaded its task after the deadline, the node has to offload to the last observed server.

4.4.4 Analysis

Figure 4.6 shows the V^* optimal threshold vs the cost associated for a processing time following uniform distribution $F(x) = \frac{1}{b-a}$ in $[0, 1]$. Figure 4.7 show the V^* optimal threshold vs the cost associated for a processing time following normal distribution $F(x) = 0.5(1 + \text{erf}(\frac{x-\mu}{\sqrt{2}\sigma}))$ with a mean $\mu = 50$ and standard deviation $\sigma = 20$; erf() is the error function. In addition, Figure 4.8 shows the V^* optimal threshold vs the cost associated for a processing time following exponential distribution $F(x) = \lambda e^{(-\lambda x)}$ with a mean $\mu = 50$. A lower cost c indicates accepting higher processing time, whereas higher cost means a high demand for a small processing time. Also, when we choose a low cost, then the offloading will happen very early at the beginning of the offloading process as the optimal threshold V^* is high.

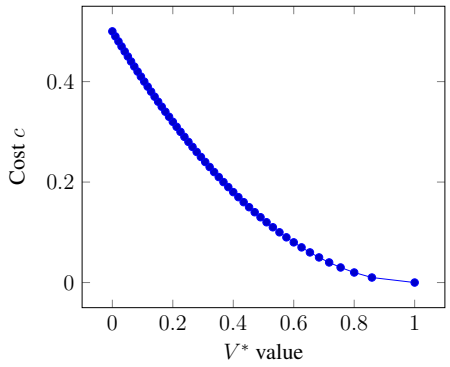


Figure 4.6: The V^* values when X is uniformly distributed in $[0, 1]$ vs. cost $c \in [0, 0.5]$.

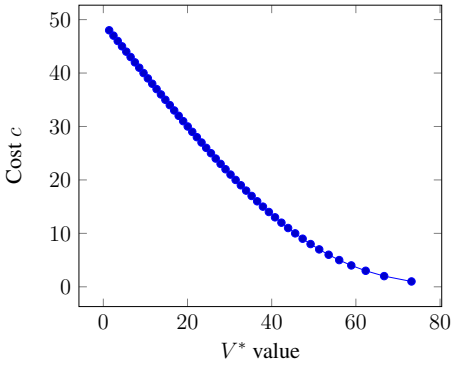


Figure 4.7: The V^* values when X is normally distributed for $\mu = 50$ and $\sigma = 20$ vs. cost $c \in [0, 50]$.

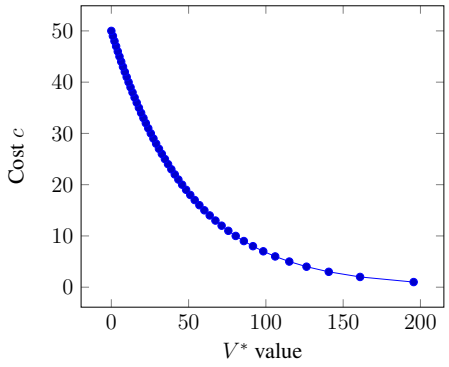


Figure 4.8: The V^* values when X is exponentially distributed for $\mu = 50$ vs. cost $c \in [1, 50]$.

4.5 Generalisation

It should be noted that the offloading decision-making proposed in Chapters 3 and 4, including the BCP, Odds, DTO and the COT, can be also applied to a situation where the mobile node is moving within the range of one MEC server and tries to choose a time instance (within a specified time horizon n) with minimised processing time. In such case, the horizon n can be divided into time slices of the same length on the scale of seconds and then we obtain the offloading rules based on the procedure of each model. The decision (which server *or* which time instance) is based on the mobility of the mobile node as well as on the density of the MEC servers deployment. For example, in AV scenario as considered in [5] and [6], we could go for MEC server selection especially if there is high mobility and high density deployment of the MEC servers. On the other hand, if the mobile node is not moving or moving in the low mobility mode, then the horizon n can be divided into decision instances with specified time horizon based on the deadline of the task to be offloaded.

4.6 Summary

In this Chapter, the objective of minimising the expected processing time is examined. Two sequential decision-making models are proposed. The optimal offloading strategies for both models are derived. To review the models, they are summarised in the Figures 4.9 and 4.10. The DTO model presented in section 4.3 and shown in Figure 4.9, takes the number of observations n and the probability distribution function $p(X_k)$ as inputs and outputs a scalar decision threshold a_k for each server $k = 1, \ldots, k$. The mobile node should offload if the observed processing time/load $X_k \leq a_k$, otherwise, the mobile node should continue observing until server n. By that time, the mobile node must offload to server n.

The COT model presented in section 4.4 and shown in Figure 4.10, takes the probability distribution function $p(X_k)$ and a cost per observation (probing cost) c as inputs and outputs a threshold V^* for each cost c. The mobile node should offload if the observed processing time $X_k \leq V^*$, otherwise, the mobile node should continue observing until a defined deadline. By that time, the mobile node must offload to the first observed server.

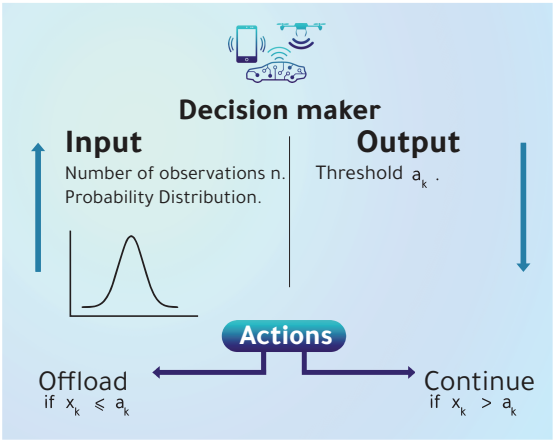


Figure 4.9: DTO.

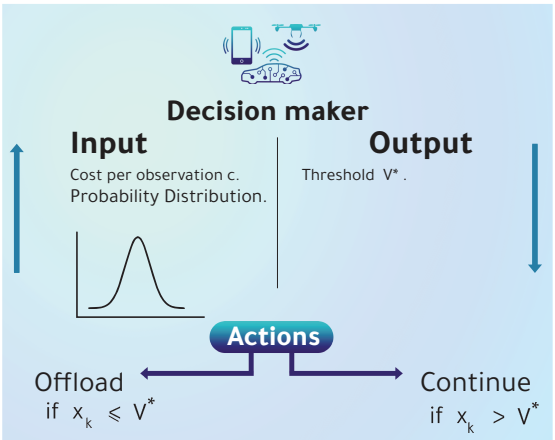


Figure 4.10: COT.

Chapter 5

Evaluation

5.1 Overview

The evaluation presented in this Chapter is to show the most important performance aspects of the proposed models in Chapters 3 and 4 and will combine simulation results, real mobility trace with real servers' data sets, and real implementation of the proposed models on real machines.

Section 5.2 shows the performance benefits of the OST-based models by generating a simulated continuous random variable based on three probability distributions, namely uniform, normal and exponential distributions. In Section 5.3, to simulate the considered environment, the two real data sets of mobility trace and real servers' CPU utilisation are combined. In Section 5.4, four machines are setup for the experiment, each of which represents a MEC server. The mobile node was represented by a process running in the machine with the highest resources.

For all the settings, an offloading decision is made based on the OST-based offloading models namely BCP (section 3.3), the quality-aware Odds model (section 3.4), DTO (section 4.3) and COT (section 4.4), the Random selection model (Random), and the p-stochastic model (p-model); which will be discussed later. The results from all models are compared with the ground truth, i.e., the Optimal model, in which the server with the minimum processing time for each offloading session is selected. The closer a model is to the Optimal, the better the model performs in terms of the task offloading decision. In the following sections, the details and the results of each setting are presented.

5.2 Simulation Based Evaluation

In this section, the proposed offloading rules are evaluated based on simulated random variable following normal, uniform and exponential distributions. The main goal of this evaluation is to see the behaviour of the models in different probability distributions. Therefore, the models and their results can be generalised to different QoS parameters with respect to the computational offloading.

5.2.1 Simulation Settings and Parameters

In the simulation evaluation, the probability distribution of the random variable X, e.g., the processing time, will be known in advance. We used `Simpy` in Python [104]; `Simpy` is a process-based discrete-event simulation framework. Each MEC server k is modelled as a resource that advertises its processing time X_k each time during the simulation. The mobile node is modelled as a process that passes by the MEC servers in 1D mobility model and checks the processing delay advertised by each MEC server. We first consider five MEC servers, i.e., $n = 5$. The processing time X is drawn from normal distribution with $\mu = 50$ ms and $\sigma = 10$ ms, uniform distribution in $[0, 1]$ and exponential distribution with $\mu = 50$. These random variables are generated using `Python` function that generates random numbers following a specified distribution. Each time, a mobile node (e.g., a vehicle) starts checking the MEC servers in sequential manner. It starts with server number 1 and applies the proposed models presented earlier to select a MEC server for offloading.

It should be noted that the processing time has been named in the literature with different terms including total delay [13], latency [47] or waiting time [6]. Also, the range of the processing time varies according to the application types. As an example, it is being ranged from 0.1 seconds to $\approx$ 800 seconds in [105] and in the range of 10 ms to $\approx$ 30 ms as in [6]. Therefore, different values or scales of X can be used with the proposed models generating similar results as we will see in the real data set experiment. Table 5.1 shows the parameters' values and range in the simulation experiment.

5.2.2 OST-Based Offloading

In the BCP, the rule when $n = 5$, based on Figure 3.2 (right), is to reject the first two servers, take the best among them as a baseline and start looking for a server that is better than the baseline. If server 5 is reached without offloading, we then must offload to server 5. In the quality-aware Odds model, we first define $\theta = 50$ (we start with the normal distribution) as a threshold. Thus, based on the proposed model, the model suggests to start from server 2

Parameter	Value / Range
X	$\mathcal{N}(50, 10)$ & $\mathcal{U}(0, 1)$ & $exp(\frac{1}{50})$
Number of mobile nodes	1000
n	$\{3, 5, 10\}$
r	$\{0, 0.25, 0.5, 1\}$
θ	$\{30, 40, 50, 60\}$ & $\{0.3, 0.4, 0.5, 0.6\}$ & $\{20, 30, 40, 50, 60\}$
c	$\{1, 2, 3, 4, 5, 20, 30\}$ & $\{0.1, 0.3, 0.4\}$ & $\{1, 10, 15, 20, 30, 40\}$
p for the p-model	0.8

Table 5.1: Simulation experiment parameters' values.

($s = 2$) and pick the server that has a processing time less than or equal to $\theta = 50$. Note that as n is set to 5, we calculate the indicators $f_1, \ldots, f_5$.

In the DTO, each server k is compared with the decision threshold a_k as proposed in Section 4.3. If the processing time X_k is less than the threshold a_k, the mobile node should offload, otherwise, it continues till it reaches the last server, and then it must offload to the last server. We first set the delay factor $r = 0$, later, the performance is shown for different r values. In the COT model (Section 4.4), there is a unique solution for V^* for each value $c > 0$. For example, in the normal distribution case, we obtain the threshold values V^* for each cost value $c \in [1, 50]$. As plotted in Figure 5.1, we can see that the generated threshold V^* values are around the mean when $c \leq 10$. In our modelling, the value of the cost c is interpreted as the need for a lower processing time, but different interpretations can be obtained based on the application requirements. As a result, we can see from Figure 5.1 that low costs have higher thresholds and thus it will accept higher processing time. In contrast, higher costs have less thresholds and thus it will look for less processing time. Once the value of c is defined, the value of V^* can be obtained as shown in Section 4.4. For example, in the normal distribution case, we start by defining the cost to be $c = 4$, and later, we show the performance for different values c. Starting from server 1, if the processing time X_k is less than that threshold V^*, we offload to that server. If we reach server 5 without offloading, we then must offload to server 5.

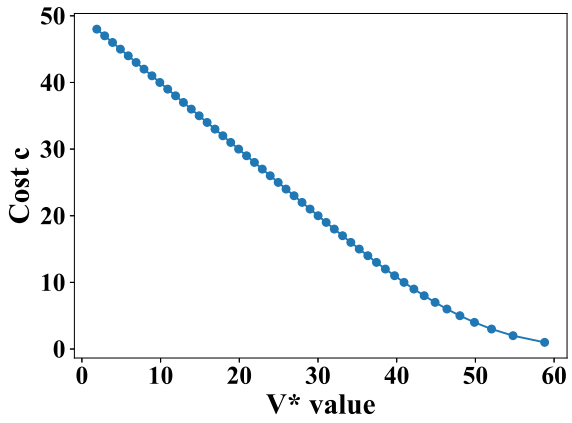


Figure 5.1: The V^* value for the processing time used in the experiment vs. cost c when X is normally distributed.

5.2.3 Baseline Models

The main approach that is being used in the literature (when dealing with offloading site selection) is the comparison with the Optimal as in [20], the Random [16, 17, 74], the nearest server (immediate offloading) as in [6, 36], and to a method belongs to the same family of the proposed algorithms in the same work as in [5, 16, 20, 45]. Similarly, in this thesis, the evaluation was limited to the comparison between the different OST models along with the Random and a probabilistic model, which we call here the p-model. These algorithms are then compared against the Optimal selection in which we select the server or time with the minimum value, e.g., processing time or CPU utilisation as we shall see in the next sections.

The reasons for taking a similar approach in this thesis are as follows: first, the core of this work is about adopting the OST in the data and task offload decision. In addition, the nature of the OST algorithms is different from other algorithms in the area. In particular, the OST algorithms are applied when incomplete information is available for the decision maker. Therefore, approaching the optimality would be a main analysis one can use when evaluating these algorithms. The optimality is a representative solution in a scenario when all the servers' information is available to the decision maker. In such a scenario, the mobile node would select the best place for offloading. Finally, these algorithms are applied in a sequential fashion which makes such algorithms difficult to compare with other algorithms.

In the considered setting in this thesis, without utilising the OST-based offloading rules, the mobile node is more likely to select the first encounter MEC server, i.e., the immediate offloading. Also, for the Edge computing offloading, such an offloading method is the most intuitive method to offload the computation tasks [36]. Thus, for this case, we utilise the p-model method as an offloading method. In the p-model, for each server, a probability of offloading is assigned; in this setting, the value of p is set to $p = 0.8$. In each user's

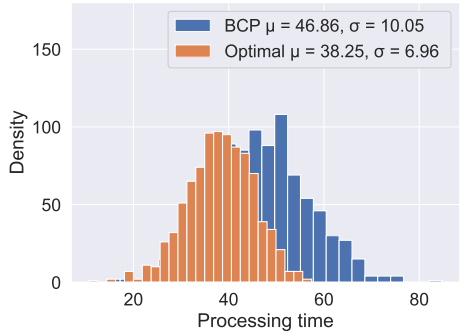
movement, each server has probability $p = 0.8$ of being selected for task offloading. More specifically, when increasing the value of p, as we do in this experiment, the probability of each server being selected for offloading is very high. As a result, the first server is more likely to be selected and is thus chosen for offloading. The p-model, therefore, is a simulation for a scenario where the mobile node offloads at the first encountered servers (the nearest Edge servers) because there is higher probability, i.e., $p = 0.8$, as we will see later in the real data set evaluation. If a server is selected, the process is stopped and the server (at that time) is selected for offloading. If there was no server selected, the last server is selected for offloading. In the Random selection model, for each user, a server, uniformly at random, is selected to offload the task. The Random is also a common method in the literature to study the algorithms performance, i.e., from an analytical point of view.

The results from all models are compared with the ground truth, i.e., the Optimal model, in which the server with the minimum processing time for each offloading session is selected. The closer a model is to the Optimal, the better the model performs in terms of the task offloading decision. The Optimal model was captured at each offloading session by selecting the server with the minimum processing time.

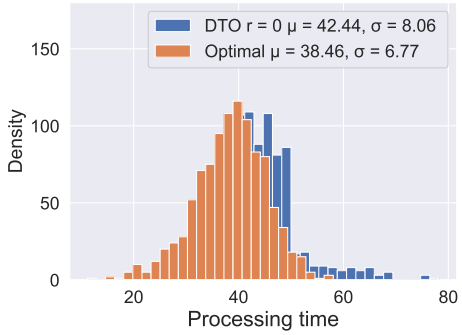
5.2.4 Results

The results of the experiment in the case where X is normally distributed are shown in Figure 5.2. We can observe that there is a noticeable overlapping between the Optimal and the OST based models (the DTO, the COT and the quality-aware Odds) as shown in Figures 5.2b, 5.2c and 5.2d, respectively. This overlapping decreases in the BCP, the Random and the p-model as it can be seen from the Figures 5.2a, 5.2e and 5.2f. However, the BCP model is achieving lower expected processing time ($\mu = 46$) than the Random ($\mu = 49$) and the p-model ($\mu = 50$). This is clear in Figure 5.3 as the difference between the Optimal and the OST based models including (BCP, DTO, COT, Odds) is significantly less than the Random and the p-model models.

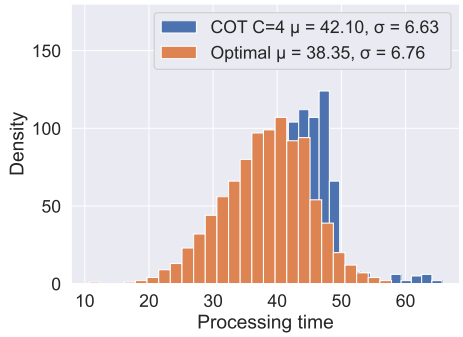
It should be noted that, in general for the DTO and the COT models, the optimal thresholds generated by each model for each observation k is close to the mean of the processing time, i.e. 50. For example, in the DTO, the generated threshold values $\{a_k\}_{k=1}^{n}$ for $n = 5$ are $\{40, 42, 43, 46, 50\}$. As we can see, the values are close to the mean of the processing time. This also applies to the COT model. Based on these optimal thresholds, we first set the threshold value $\theta = 50$ for the Odds, and later we show the performance for different θ values.



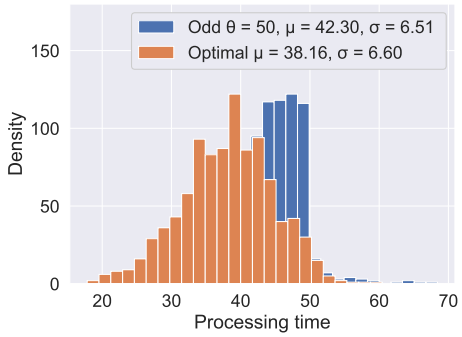
(a) BCP and the Optimal selections.



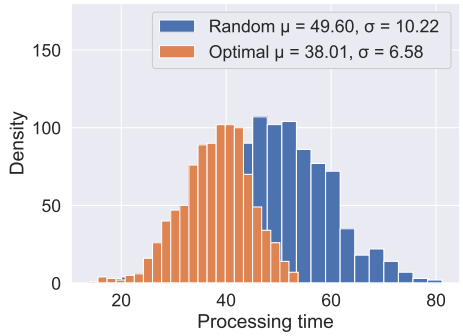
(b) DTO and the Optimal selections.



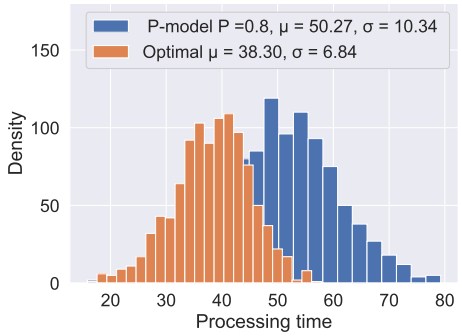
(c) COT and the Optimal selections.



(d) Odds and the Optimal selections.



(e) Random and Optimal selections.



(f) P-model and Optimal selections.

Figure 5.2: Simulation results for all the models when X normally distributed.

As it can be seen from the Figures, we had a good performance in the Odds. This good performance is due to the fact that we have around 42% probability of picking a server with processing time less than 50[1]. Therefore, a lesson learnt here is that setting a threshold value close to the mean value achieves less processing time. We had better performance in the BCP model than the Random and the p-model as the BCP offloading policy has higher probability of offloading to the minimum processing time than the Random and the p-model as we observed earlier in Section 3.3. This higher probability is translated into less expected processing time than the Random and the p-model. We should note that, we have similar

[1]This probability is calculated using equation (3.12).

probability of offloading the best in the BCP and the Odds, but having a defined threshold θ when checking the MEC server has increased the performance in the quality-aware Odds model.

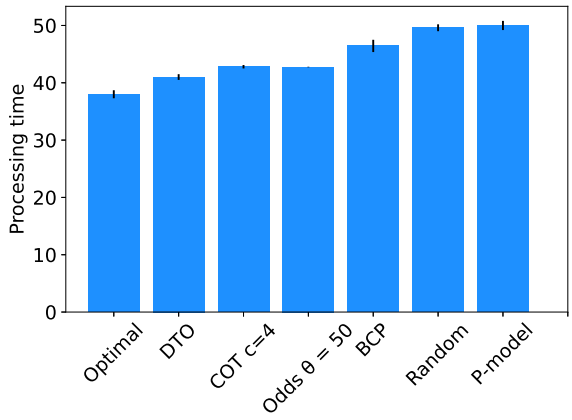
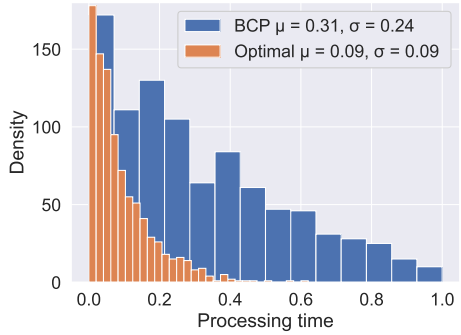
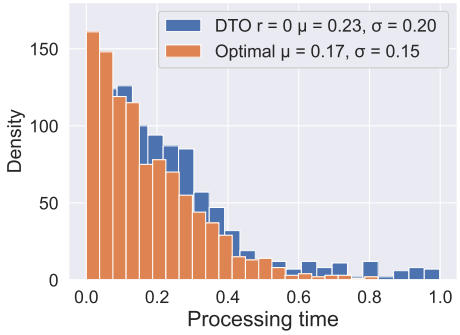


Figure 5.3: Confidence interval in the simulation experiment when X is normally distributed.

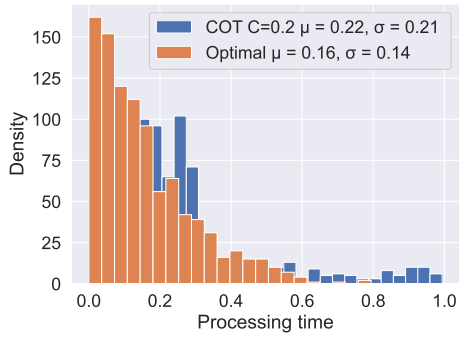
In the previous experiment, the random variable X to be observed and optimised is following normal distribution. Now, X is uniformly distributed scaled in $[0, 1]$. This can refer to the server utilisation, i.e., the CPU utilisation [106]. For example, 0.5 indicates that 50% of servers' CPU is utilised. The same steps, as in the previous experiment for all the models, are followed. In the DTO model, the delay factor is first set to $r = 0$, and later we show the results for the rest of r values. In the COT model, the optimal threshold V^* values for each cost value $c \in \{0.1, 0.2, 0.3, 0.4\}$ are obtained. The results when $c = 0.2$, where the optimal threshold $V^* \approx 0.36$, is firstly shown. Later, the performance for the rest of the values c will be presented. The interpretation for the cost is similar to the situation when we have X normally distributed, i.e., higher cost (small threshold V^*) means high demand for less processing time. In the quality-aware Odds model, the threshold θ is set to $\theta = 0.5$, thus, there is around 42% chance of offloading to a server with $X \leq \theta = 0.5$. Although we have the same probability in the BCP model, but again, it turns out that setting a threshold can improve the performance of the Odds model. In general, we can see from the results shown in Figure 5.4 and Figure 5.5 that the models performance is similar to the results we obtain when X is following normal distribution. We still have the DTO, COT models perform the best, and in general, the OST based models are closer to the Optimal than the Random and $p-$model.



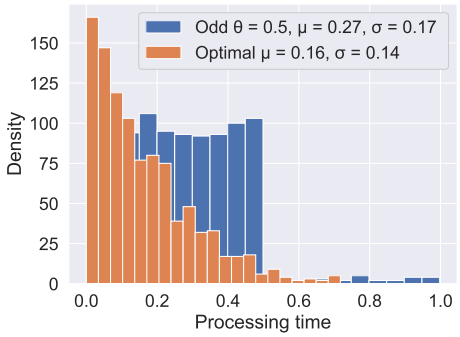
(a) BCP and the Optimal selections.



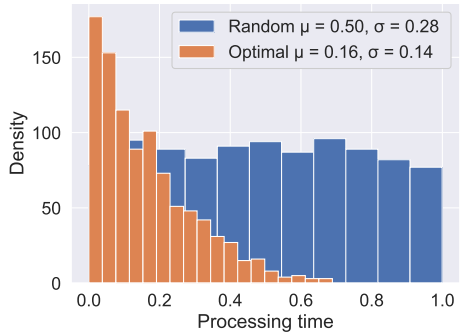
(b) DTO and the Optimal selections.



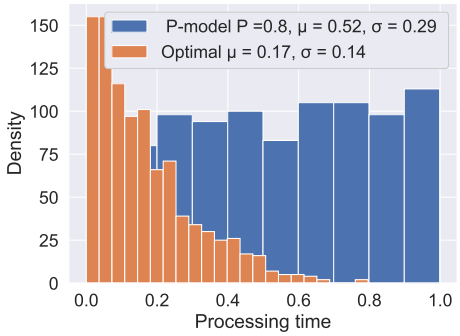
(c) COT and the Optimal selections.



(d) Odds and the Optimal selections.



(e) Random and Optimal selections.



(f) P-model and Optimal selections.

Figure 5.4: Simulation results for all the models when X uniformly distributed.

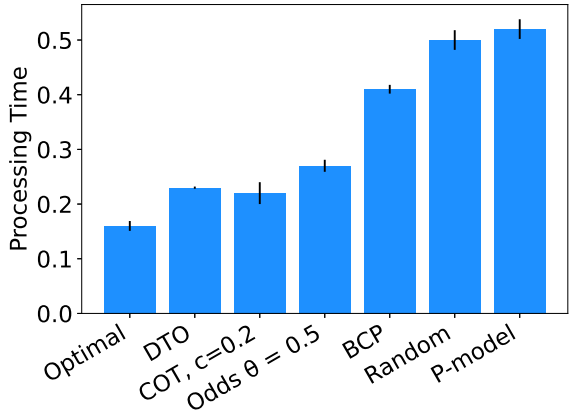
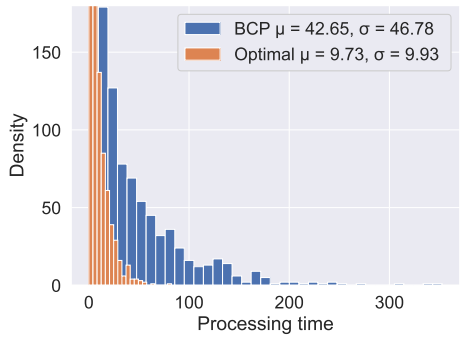
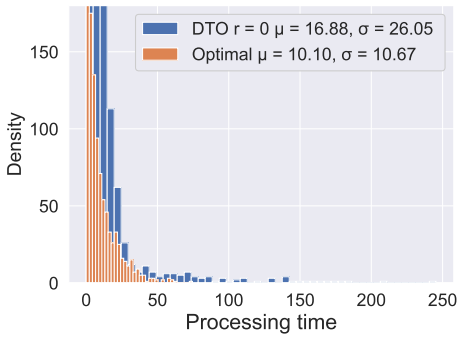


Figure 5.5: Confidence interval in the simulation experiment when X uniformly distributed.

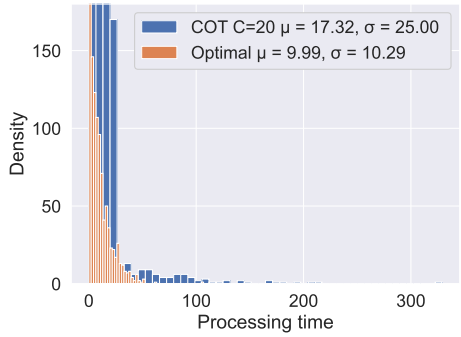
In addition to the normal and uniform distributions, the exponential distribution is considered in this experiment with $\mu = 50$. The steps that were carried out in the previous distributions are also followed in this experiment. In the DTO model, the delay factor is first set to $r = 0$, and later we show the results for the rest of r values. The values of $\{a_k\}_{k=1}^{n}$ are obtained as shown in Figure 4.4. In the COT model, the optimal threshold V^* values for each cost value $c \in [1, 50]$ are obtained as shown in Figure. 4.8 presented earlier in Chapter 4. We first consider the case where $c = 20$ where the optimal threshold $V^* \approx 45.81$ as it achieves the lowest expected simulated X among other V^* values. Later, the performance for different values c will be presented. The interpretation for the cost is similar to the situation when we have X normally and uniformly distributed, i.e., higher cost (small threshold V^*) means high demand for less processing time. In the quality-aware Odds model, the threshold θ is set to $\theta = 50$, thus, there is around 44% chance of offloading to a server with $X \leq \theta = 50$. In general, we can see from the results shown in Figure 5.6 and Figure 5.7 that the models performance is similar to the results we obtain when X is following normal and uniform distributions. We still have the DTO model performs the best, and in general, the OST based models are closer to the Optimal than the Random and $p-$model. However, the performance of the BCP decreases as it can be seen in Figure 5.6a and the difference between the Optimal and the BCP is higher than the case of the normal and the uniform distributions.



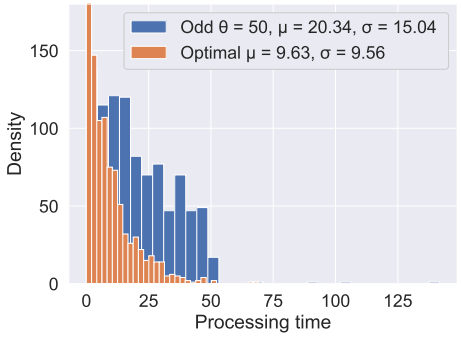
(a) BCP and the Optimal selections.



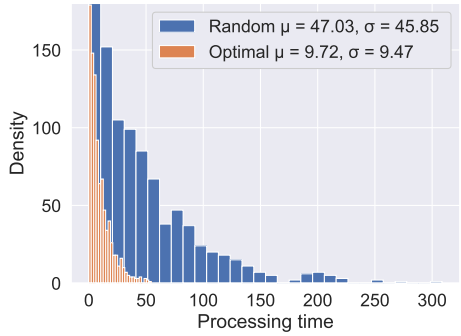
(b) DTO and the Optimal selections.



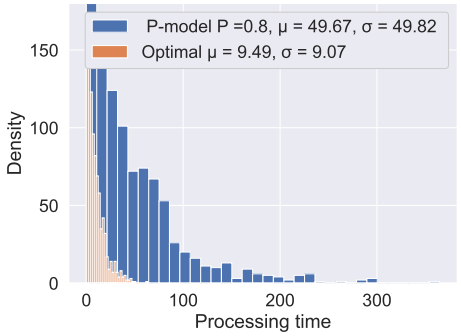
(c) COT and the Optimal selections.



(d) Odds and the Optimal selections.



(e) Random and Optimal selections.



(f) P-model and Optimal selections.

Figure 5.6: Simulation results for all the models when X exponentially distributed.

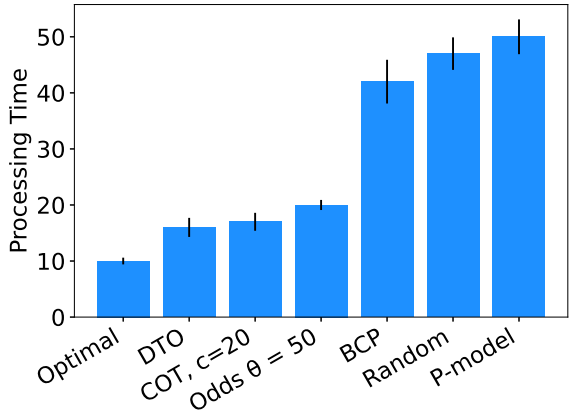


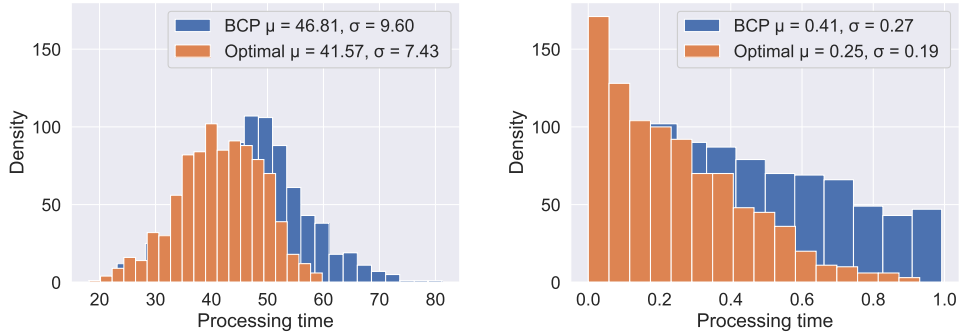
Figure 5.7: Confidence interval in the simulation experiment when X exponentially distributed.

5.2.5 Sensitivity Analysis (Simulated Environment)

In this subsection, the models' results for different parameters values are investigated further. This includes: the results of the BCP model (Section 3.3) for different values of n, the DTO model (Section 4.3) for different delay factors r, the COT model (Section 4.4) for different costs c, and the quality-aware Odds model (Section 3.4) with different θ values.

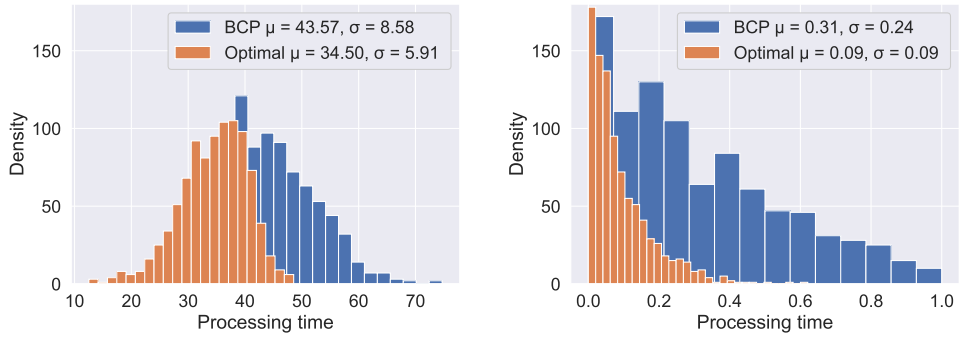
5.2.5.1 BCP Model Analysis

We observe that when n is small, the difference between the BCP and the Optimal decreases. For normally distributed X, the difference was 5.24, and 9.07 for $n = 3$ and $n = 10$ respectively as shown in the Figures 5.8a and 5.8c. This is also true when X uniformly distributed: the difference was 0.16, and 0.22 for $n = 3$, $n = 10$ respectively as shown in the Figures 5.8b and 5.8d. These results support Figure 3.2 (left), i.e., the probability of offloading to the best decreases and approaches 36% as n increases.



(a) BCP when $n = 3$ and X normally distributed.

(b) BCP when $n = 3$ and X uniformly distributed.



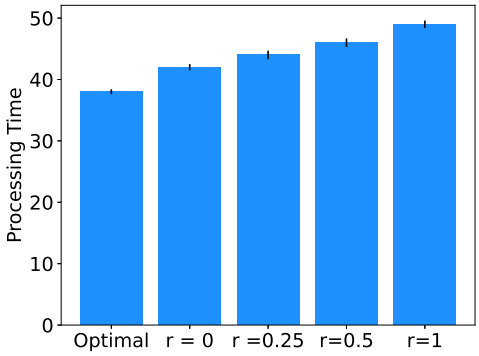
(c) BCP when $n = 10$ and X normally distributed.

(d) BCP when $n = 10$ and X uniformly distributed.

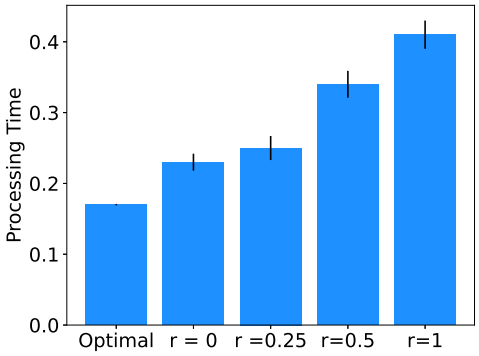
Figure 5.8: BCP models for different values n.

5.2.5.2 DTO Model Analysis

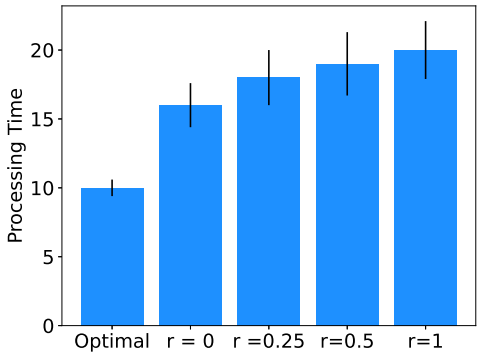
In the DTO model, a higher value r generates small thresholds $\{a_k\}_{k=1}^{n}$. As a result, in the case of higher values r, the model will delay the offloading trying to find processing time less than very small thresholds $\in \{a_k\}_{k=1}^{n}$. On the other hand, when r is small, the model generates higher values of $\{a_k\}_{k=1}^{n}$ (close to the mean); which enforce the decision maker to offload earlier than the case when r higher. These cases can be seen in Figure 4.2, 4.3 and 4.4 in Chapter 4. In general, the performance of the DTO was the best when $r = 0$. We see from Figure 5.9 that as we increase the value of r, the model tends to have higher difference compared to the Optimal selection for the all the considered distributions.



(a) Confidence interval when X normally distributed.



(b) Confidence interval when X uniformly distributed.



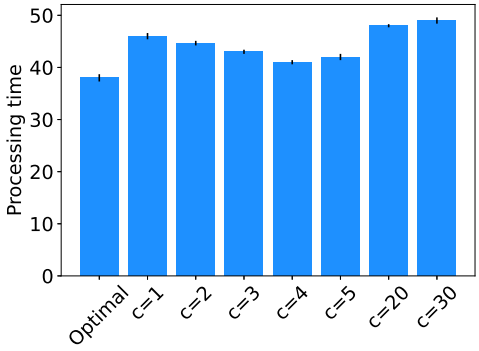
(c) Confidence interval when X exponentially distributed.

Figure 5.9: Confidence interval in the DTO for different r values.

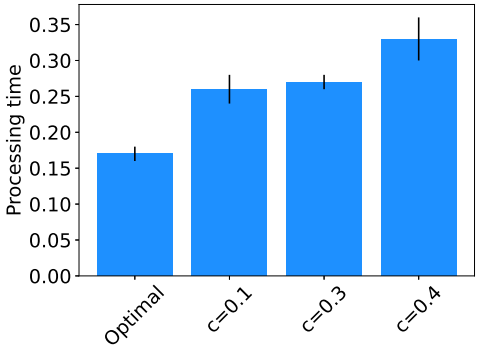
5.2.5.3 COT Model Analysis

In our modelling, high value c will generate small threshold V^*. This can be interpreted as the high demand for small value of X. On the other hand, small value of c will generate large threshold V^*. Therefore, when we set the cost to a small value, then, it implies we are tolerant to expect higher value X. Figure 5.10 shows the confidence interval and the average processing time achieved by the COT model for different c values when X is normally distributed (Figure 5.10a), when X uniformly distributed (Figure 5.10b), and when X is exponentially distributed (Figure 5.10c). In the normal distribution case, we can see from the results that the processing time achieved by the COT is higher when $c = 1, c = 20, c = 30$. When $c = 1$, V^* is high, thus, the mobile node offloads at the firs server. When $c = 20, c = 30$, V^* is very small, thus, it delays the offloading and in fact the mobile node did not find a server with processing time less than V^*. Therefore, the mobile node has to offload to the last server. The value of V^* was around the mean when $c = 3, 4, 5$. Thus, we had a closer processing time to the Optimal. This is also true for the uniform distribution, when $c = 0.1$, the value of the V^* was 0.5. Therefore, in the case of the normal and uniform distributions, we observe that when the threshold V^* is close to the expected value (mean), the COT performs

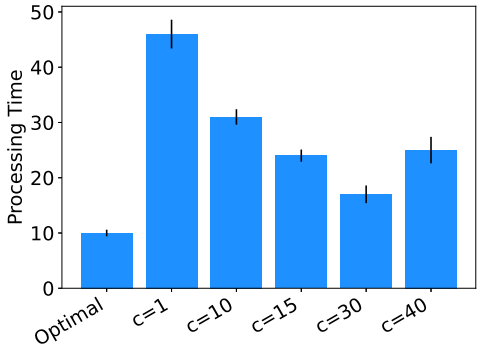
better. When X is exponentially distributed, however, we had different behaviour for the COT model. The model performs better when $c = 30$ with V^* value of ≈ 26 which is not that close to the mean. There is a good performance when we have $c = 15$ and $c = 40$ with values of $V^* = 60, 11$ respectively.



(a) Confidence interval when X normally distributed.



(b) Confidence interval when X uniformly distributed.



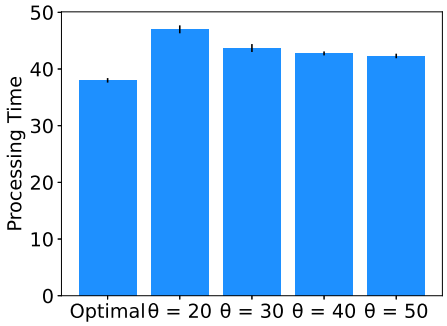
(c) Confidence interval when X exponentially distributed.

Figure 5.10: Confidence interval in the COT for different cost c values.

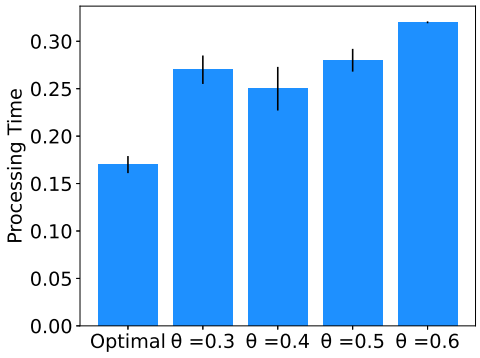
5.2.5.4 Odds Model Analysis

Figure 5.11 shows the confidence interval and the average processing time achieved by the quality-aware Odds model for different θ values when X is normally distributed (Figure 5.11a), when X is uniformly distributed (Figure 5.11b), and when X is exponentially distributed (Figure 5.11c). As discussed in Chapter 3, a main component of the proposed Odds model is to define a threshold (the mobile node is looking for less than this threshold) which is then compared with each observation. This might refer to different types of requirements, e.g., processing time or load. In the normal distribution case, the model manages to meet the threshold when $\theta = 50$, but it fails in the rest of the thresholds. The overall performance was similar to the COT model; we observe that when θ is close to the mean, the model performs better. As mentioned earlier, when $\theta = 50$, we have around 42% chance of offloading to a

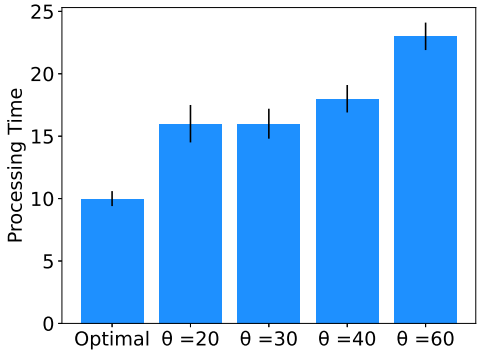
server with processing time less than or equal to 50. This is also true when $\theta = 60$, i.e., we have around 42% chance of offloading to a server with processing time less than or equal to 60. However, due to the higher value of the threshold, i.e., 60, we had higher processing time than the processing time when we set θ to 50. Setting the value of θ to 40 had an acceptable performance. We had higher processing time when $\theta = 30$ as we have small chance of offloading to a server with the specified threshold.



(a) Confidence interval when X normally distributed.



(b) Confidence interval when X uniformly distributed.



(c) Confidence interval when X exponentially distributed.

Figure 5.11: Confidence interval in the quality-aware Odds for different θ values.

The model manages to meet all thresholds when X is uniformly distributed. However, the overall performance was different in this case. When $\theta = 0.4, 0.5$ and 0.6, we have a higher chance of offloading to a server meeting the specified thresholds, i.e., $> 40\%$. However, the performance was not good when $\theta = 0.6$. The reason for having a higher processing time is because of the higher value of threshold (0.6), and thus, the mobile node will accept a server with a higher processing time. We had an improvement when $\theta = 0.4$ over the case when $\theta = 0.5$. The reason for this improvement is that the model when $\theta = 0.4$ starts the search at index 1, i.e., $s = 1$ [2], thus, this index allows the model to have a larger search space over

[2]The value of s is the index where the Odds reach or exceed the value of 1 for the first time, please see subsection 3.4.1 for more details.

the case when $\theta = 0.5$ as the model starts at index $s = 2$. The model manages to meet all thresholds when X is exponentially distributed. The minimum processing time was when $\theta = 20, 30$ which is less than the mean by more than 10.

Therefore, the main observation for the Odds model when X is $\mathcal{N}(50, 10)$ or $\mathcal{U}(0, 1)$ is that setting the threshold θ to the expected value or less by ≈ 10 achieves less processing time. In contrast, when X is exponentially distributed, the minimum processing time was when $\theta = 20, 30$ which is less than the mean by more than 10. In this case, the probability of meeting all the thresholds are still higher than 40%, but, again, it turns out that setting smaller thresholds results in less X. Also, the value of s (the index of where the mobile should start accept server for offloading) has an impact in the overall performance.

Based on this observation, there is an upper limit and a lower limit for the threshold θ to be closer to the Optimal. For example, the upper limit when X is $\mathcal{U}(0, 1)$ has been tested, i.e., when the value of $\theta = 0.6$. When compared to the Optimal, the difference is significant, indicating that if you try a higher threshold value, e.g., $\theta > 0.6$, the performance would decreases, therefore there is no need to test higher thresholds. For the lower limit, we have started testing the threshold by testing $\theta = 0.4$ and we noticed that there is an improvement over $\theta = 0.5$ and thus we continue comparing till we observed that when $\theta = 0.3$, the performance is not getting improved and therefore, trying for example less threshold ($\theta = 0.2$) is going to be worst than the case when $\theta = 0.3$ and $\theta = 0.4$.

In general, the interpretation for this is that when the threshold is high, there is a high chance of accepting higher observation and thus resulting in a higher selected processing time. On the other hand, when the threshold is small, the model will not find an observation meeting the very small threshold. Thus, the model will keep looking for a value and it will lead to pick the last one as the chance of finding a small value for the random variable is small. In other words, the model will always take the last observation and that is why we had higher processing time in the case of $\theta = 0.3$, for example, in the uniform distribution. We should note that each θ value has different stopping index s based on model presented in Section 3.4. As an example, when X is normally distributed and $\theta = 30, 40$, $s = 1$, and when $\theta = 60$, $s = 3$. When X is uniformly distributed, e.g. when $\theta = 0.3$, $s = 1$, and when $\theta = 0.6$, $s = 2$. When X is exponentially distributed and $\theta = 60, 40$, for example, $s = 2$, and when $\theta = 30, 20$, $s = 1$.

To summarise, in general, the models DTO, COT and the Odds are showing a good performance and their selection are higher than the Optimal by 10% in the normal distribution, $\approx 35\%$ in the uniform and $\approx 68\%$ in the exponential distribution. The BCP model is still performing better than the first selection and the Random model.

5.3 Real Data Sets Based Evaluation

In this section, real data sets to evaluate the proposed models are also utilised. The purpose of this evaluation is to see how the OST-based models perform when dealing with real data sets. The random variable in this evaluation is represented by CPU utilisation and the aim is to offload to a server that has less CPU utilisation.

5.3.1 Data Sets

To simulate the movements of the mobile nodes, first, the real data set of taxi cabs' movements in Rome [107] is used. The data set contains GPS coordinates of 320 taxis collected over 30 days. Each row in this data set has the cab-id, date/time and GPS coordinates of the current location. It is worthwhile to mention that the use of mobility trace here is not for studying the mobility of users. It is used in this experiment to use each time movement as location or time to check for a server or a time to offload. In other words, each movement is modelled as an observation or connection to a MEC server. Figure 5.12 shows the movements of the cars in Rome map.

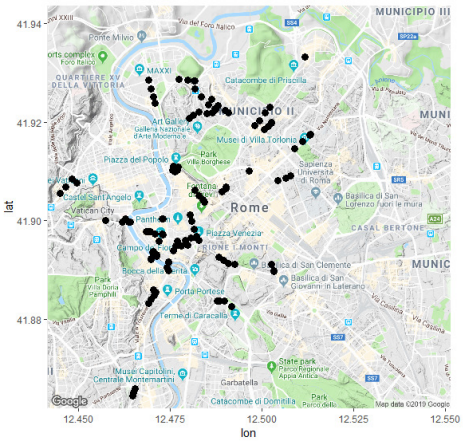


Figure 5.12: Taxis trajectories in Rome.

The processing time is represented in this experiment by real servers' utilisation (the CPU utilisation) obtained from [108]. In the servers' data set, there are around 150 servers' data (more than 1 billion rows). Thus, for each movement, the car picks a server from the servers' data set, checks that server utilisation and takes a decision of whether the car should offload at that time or continue observing based on the decision suggested by the model as explained earlier in the simulation evaluation section. The movements of over 5 days (5000 rows of movements) are used in the experiment. An offloading decision was taken for each minute. Thus, we have more than 1000 offloading decisions. This will ensure to see the behaviour of

the proposed models for a long time. Figure 5.13 shows the probability distribution of the servers' utilisation for the all servers in the data set. We can see that the servers' utilisation in general follows normal distribution with $\mu = 36$ and $\sigma = 16$. Also, for illustrative purposes, an example of one offloading decision session is shown in Table 5.2. In Table 5.3, we show the key parameters' values in this experiment.

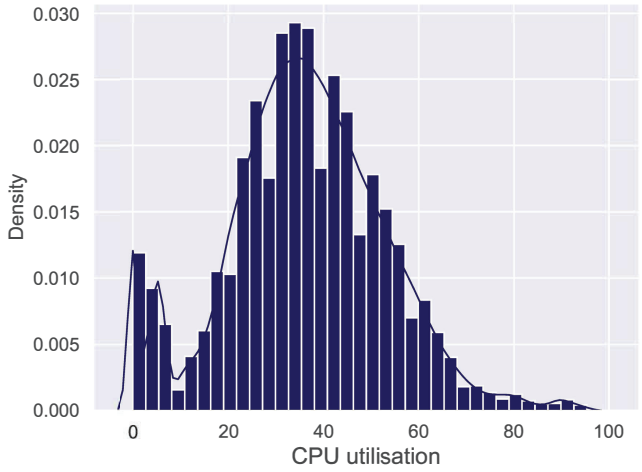


Figure 5.13: The distribution of the servers' CPU utilisation.

Cap id	Movement time	Location	Machine name	CPU utilisation
156	2014-02-05 00:11:01	(41.8911, 12.49073)	m_1939	(51)
156	2014-02-05 00:11:11	(41.89905,12.4899)	m_1936	(47)
156	2014-02-05 00:11:22	(41.8994,12.48940)	m_1941	(20)
156	2014-02-05 00:11:31	(41.8994,12.489401)	m_1941	(37)

Table 5.2: A sample of the data set used in the experiment.

Note that, as the server utilisation is approximately following normal distribution, when we apply the OST models, we have to feed the models (Odds, DTO and COT) with the mean and the standard deviation. In this experiment, the mean and the standard deviation were taken once at the beginning of the experiment for the whole servers' utilisation data set. In other words, the models are not applied with the mean and the standard deviation of the observed utilisation during the experiment. Instead, this information is only taken once when the experiment starts. This is an important aspect of conducting this experiment as in real world scenario, the mobile node does not know the mean and the standard deviation of a specific MEC server, but can obtain this information from historical data for the MEC servers in one area in specific time with the help of MEC servers operators.

Similar to the simulation setting, the results from all models are compared with the ground truth, i.e., the Optimal model, in which the server with the minimum CPU utilisation for each

Parameter	Value / Range
X	real servers CPU utilisation in $\mathcal{N}(36, 16)$
Number of movements	5000 movements
Number of offloading decision	> 1000
n	5
r	$\{0, 0.25, 0.5, 1\}$
θ	$\{20, 30, 40, 50, 60\}$
c	$\{1, 2, 3, 4, 5, 20, 30\}$
p for the p-model	0.8

Table 5.3: Real data set experiment parameters' values.

offloading decision session is selected. For example, the Optimal in Table 5.2 is to offload at 00:11:22 with CPU utilisation of 20%. The closer a model is to the Optimal, the better the model performs in terms of the task offloading decision. All models are applied on each minute (offloading decision session) for evaluation. In short, each minute consists of around 5 movements. Each model selects a server for offloading as suggested by that model. Then, the average server utilisation achieved by each model in all offloading decision session is taken.

5.3.2 Results

Figure 5.14 shows the average server utilisation suggested by each model. We can see that the OST models are the closest models to the Optimal. The DTO performs better than the rest of the models with absolute difference, compared to the Optimal of 5 and it is higher than the Optimal by 23% as shown in Figures 5.15a and 5.15b.

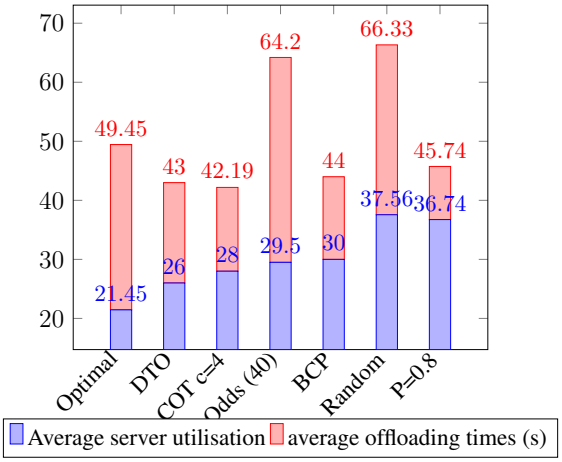
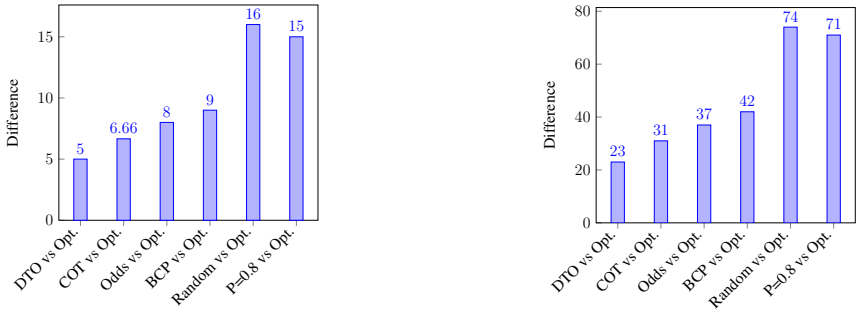


Figure 5.14: Average CPU utilisation and average offloading times suggested by each model.

In reality, the mobile node would normally offload to the first server or in the first time. A simulation for such case is the p-model with $p = 0.8$. This is clear in Figure 5.14 where the



(a) Absolute difference.

(b) Percentage difference.

Figure 5.15: Difference between the Optimal and all models.

p-model has the lowest offloading times (offload earlier than other models). We can see from the results that the p-model is too far from the Optimal. In other words, our results show that going with the first server (time) or (immediate server) is not a good idea. Moreover, which server or time is the Optimal is not known and not provided to the mobile node. In other words, in the considered environment, the Optimal is not available to the mobile node so having the OST-based model implemented in the mobile node can achieve near-Optimal server utilisation.

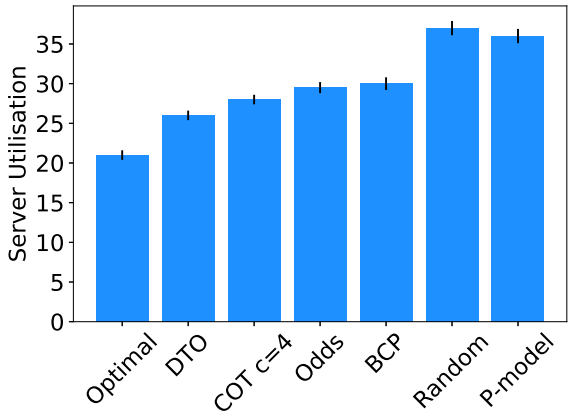


Figure 5.16: Confidence interval for the the real data sets experiment.

5.3.3 Sensitivity Analysis (Real Data Sets)

To further investigate the difference between the proposed models and the Optimal, the confidence intervals (95% confidence limits) for the results obtained by each model are plotted in Figure 5.16. We can see that the difference is more significant with the Random and p-model. Furthermore, the difference between the server utilisation achieved by the Optimal and the server utilisation achieved by the BCP is greater than the rest of the models (DTO,COT, and Odds). The DTO model, on the other hand, is the closest model to the Optimal.

5.3.3.1 DTO Model Analysis

In the DTO model, a higher value r produces small thresholds $\{a_k\}_{k=1}^{n}$, just as it did in the simulation experiment. As a consequence, when r is large, the model will postpone the offloading in order to find processing time below very small thresholds $\{a_k\}_{k=1}^{n}$. When r is small, however, the model produces higher values of $\{a_k\}_{k=1}^{n}$ (close to the mean), forcing the decision-maker to offload sooner than when r is higher. Similar to the situation we had in the simulation experiment, in general, the performance of the DTO was the best when $r = 0$. We see from Figure 5.17 that as we increase the value of r, the model tends to have higher difference compared to the Optimal selection.

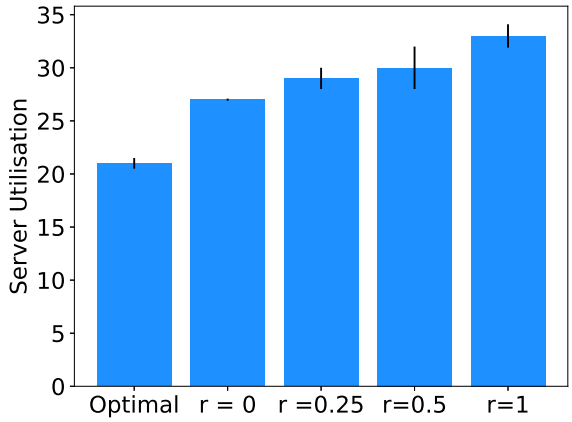


Figure 5.17: Confidence interval in the DTO for different r values.

5.3.3.2 COT Model Analysis

Figure 5.18 shows the V^* optimal threshold for the COT model vs the associated cost for the server utilisation used in the experiment. Similar to simulation experiment, a lower cost c indicates accepting higher server utilisation, whereas higher cost means a high demand for small server utilisation.

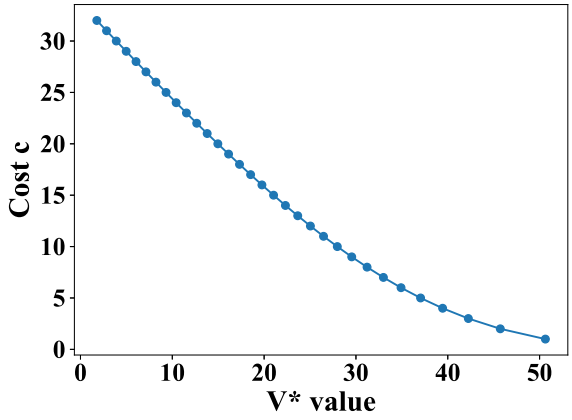


Figure 5.18: The V^* value for the server utilisation used in the experiment vs. cost c.

Figure 5.19 shows the confident interval and the average CPU utilisation achieved by the COT model for different cost c values. We observe that the server utilisation is closer to the Optimal when $c = 3$, $c = 4$ and $c = 5$. The V^* optimal threshold values when $c = 3$, $c = 4$ and $c = 5$ are 42, 39 and 37 respectively which are around the mean of the server utilisation.

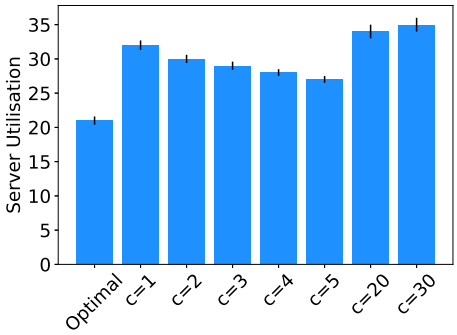


Figure 5.19: Confidence interval in the COT for different cost c values.

5.3.3.3 Odds Model Analysis

Similarly, Figure 5.20 shows the confident interval and the average server utilisation achieved by the Odds model for different θ values. As discussed in the simulation experiment, a main component of the proposed Odds model is to define a threshold (the mobile node is looking for less than this threshold) which is then compared with each observation. In this experiment, this refers to the requirement for a server with CPU utilisation less than specific threshold. Different values of the threshold θ are being tested, i.e., $\theta \in \{20, 30, 40, 50, 60\}$. As it can be seen from Figure 5.20, the model manages to have server utilisation less than the specified thresholds except the case when $\theta = 20$. We also observe that when the threshold is close to the mean $(\mu \approx 36)$, the server utilisation gets closer to the Optimal. This is a result of

these values being closer to the mean ($\mu = 36$) with higher probability of success. When the $\theta = 30$, the model performs better than the rest. The interpretation for this is that this value is close to the mean and also the index s in which the mobile node should start accepting MEC server is $s = 1$, which gives the model higher chances to improve the performance. When $\theta = 40$, however, the value of s is $s = 2$ which has less search space than the case when $\theta = 30$. When $\theta = 20$, the model will not find an observation meeting the very small threshold. The model will keep looking for a value and it will lead to pick the last one as the chance of finding small value for the random variable is small. As a result, the model will always take the last observation and that is why we had high value in this case. When the threshold is high as the case when $\theta = 60$, the model keeps the server utilisation less than 60, but the performance was less than the rest of the model. The interpretation for this case is that as the threshold is high, there is higher chance of accepting the first observation and thus resulting in a higher selected value. This supports our findings in the simulation experiment.

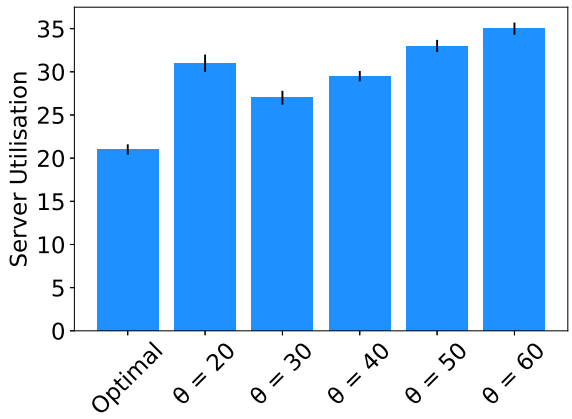


Figure 5.20: Confidence interval in the Odds model for different θ values.

5.3.3.4 Number of Successful Offloading

In addition to the average server utilisation as an performance metric, we use the number of successful offloading for each model. The number of successful offloading refers to number of offloading decisions, suggested by each model, that meets specific requirements. To have an idea about the number of successful offloading metric for each model, let's first assume that we have 3 different MEC applications x, y and z. Each of which has a specific requirement. For example, application x requires a CPU utilisation less than 10, application y requires a CPU utilisation less than 20 and application y is tolerant to offload to a server with CPU utilisation less than 30. Now, if an offloading happens to server with utilisation less than 10, we then consider that a successful offloading for application x.

Figure 5.21 shows the number of successful offloading for different resource requirements

for all the models. For an application that requires, $\leq 10\%$ CPU utilisation, the Optimal achieves 102 successful offloadings, i.e., 102 times the Optimal selects a server with a utilisation less than 10%. In the second requirements, $\leq 20\%$, the Optimal had 463 successful offloadings. For the first and the second requirements, the Odds model was the best among the other models. In the third requirement, $\leq 30\%$, the Optimal had 887 successes. Again, the Odds model was the closest one to the Optimal in the number of successful offloading with 797 success.

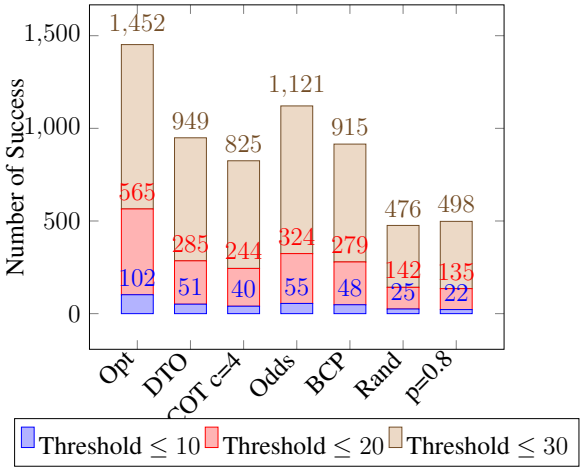


Figure 5.21: The number of successful offloadings for each model based on different threshold values.

5.4 Real Environment Evaluation

In this section, the proposed offloading rules are implemented in real machines with a focus on the processing time (execution time) of a real world computing task. The set-up of the experiment, including the machines used and the computing task, are described, followed by the results for each model.

5.4.1 Machines

Recall that, in our setting, there is a set of servers and a client (a mobile node). To emulate such a setting, three machines and one virtualised machine are used, equivalent to four servers acting as a place to offload the computing task. The client was emulated as a process in one of these machines. Each machine runs a server Python socket code and waits for the client to connect if that machine is selected. A client Python code is also executed in the client machine. The details of the machines are shown in Figure 5.22.

Clearly, there are differences in terms of the devices' capabilities in terms of the CPUs and RAMs, and these differences can be a representation for the servers having different loads

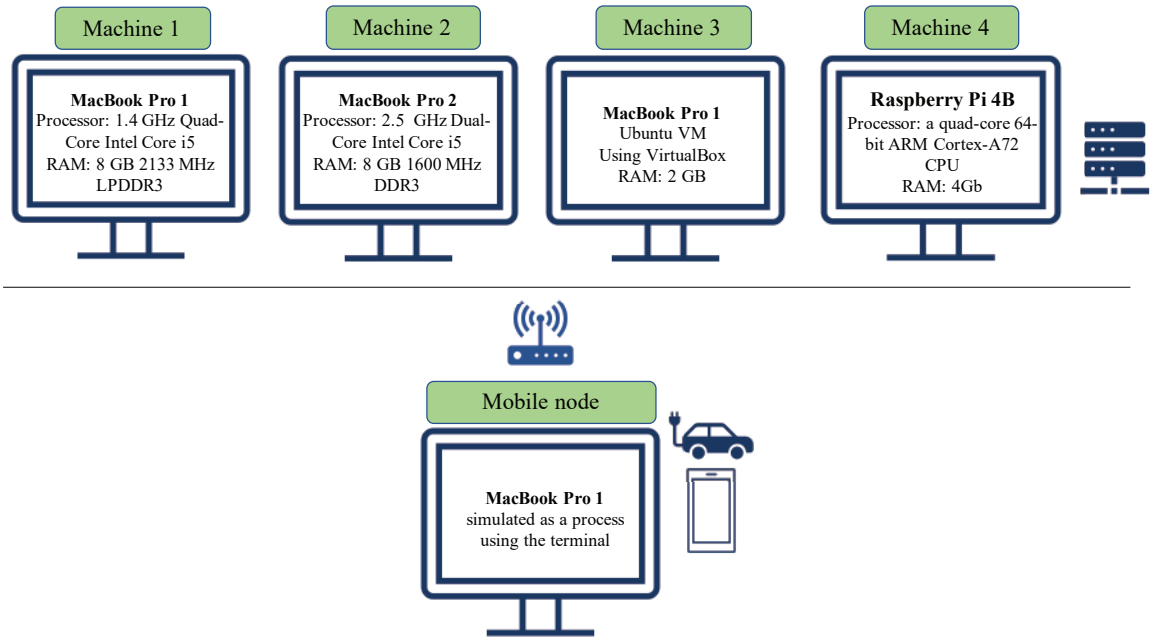
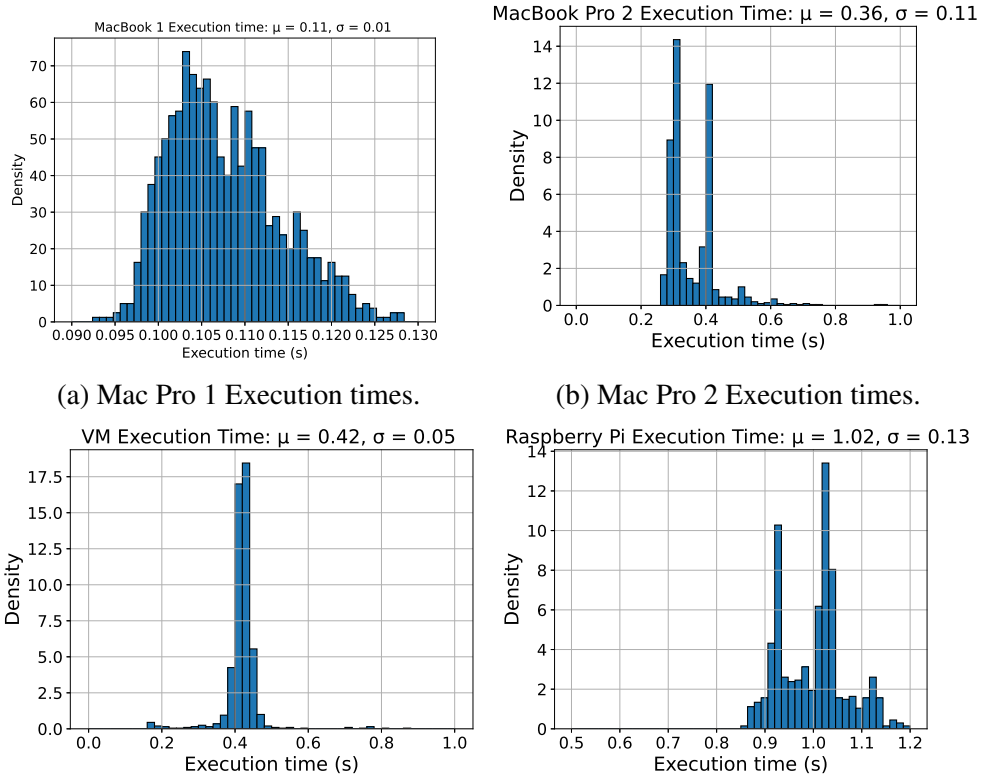


Figure 5.22: Machines used in the experiment.

over time. Each machine is assigned with a number, i.e., (1) for MacBook Pro1, (2) for MacBook Pro2, (3) the virtualised machine and (4) the Raspberry Pi, based on the average execution time for each machine, and these numbers are stored in a list $[1, 2, 3, 4]$. The random variable used to take the decision is the average execution time for the task over the long run (when running the task more than 1000 times) as shown in more detail below.

For the considered setting in this thesis, the mobile node is moving and observing these servers. Therefore, at each time or each decision episode, the order of the list is shuffled using the `shuffle` function in Python [109] so that the mobile node is trying all different combinations of order. Then, based on the average execution time for the task in each machine, the decision is made. For example, in the simplest model, i.e., the BCP model presented in Section 3.3, the machines are initially listed as $[1, 2, 3, 4]$; the list is then shuffled and the process of selecting a machine for offloading begins.

As an example, say the list is [2,1,3,4] after shuffling, then the first selection (here it is 2 == MacBook Pro 2) as a baseline which is then compared with the rest. The selection, based on the BCP model, is to take the second option as it is better than the baseline; i.e., the average execution time of 1==MacBook Pro 1 is better than that of the 2==MacBook Pro 2 as shown in Figure 5.23a and Figure 5.23b. In the threshold OST based models, i.e., DTO and COT, each selected average execution time is compared with the threshold generated by the model. The probability distribution of each machine with the average execution time for the task (explained below) are shown in the Figures 5.23a, 5.23b, 5.23c and 5.23d.



(a) Mac Pro 1 Execution times.

(b) Mac Pro 2 Execution times.

(c) Virtual machine.

(d) Raspberry Pi machine.

Figure 5.23: The distribution of the execution times for the machines used in the experiment.

5.4.2 Task

The task to be offloaded is an image recognition task in which the client has an image that must be identified. The client sends the image to the server which then does the recognition (prediction) and returns a list of string of the possible answers with probabilities being assigned to each prediction. A machine learning Python-trained library (`imageai Prediction`) has been adopted [110].

5.4.3 Metrics and Analysis

The execution time is the time between the connection being established (when the client starts sending the image) to the point at which the client receives the results from the server, and the experiment is to see if this time can be minimised by applying the OST-based models. As the devices are connecting to the same router, the effect of the network link is negligible. This was checked by running the client code in the Raspberry pi machine , and this generated similar results to running the client from MacBook Pro 1 as shown in Figure 5.23d and 5.24.

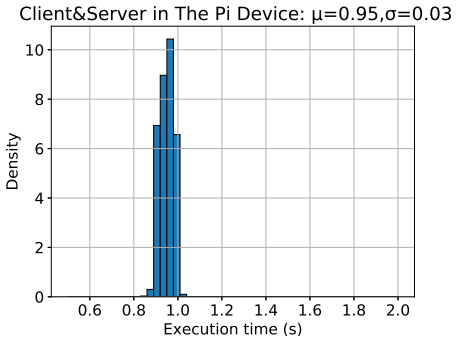
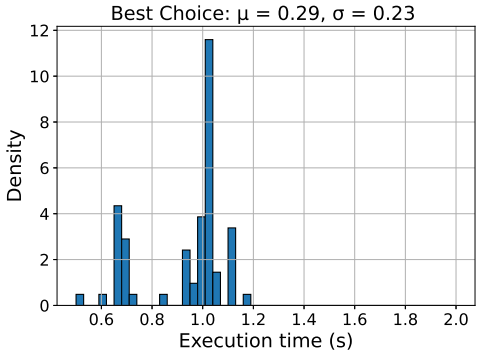


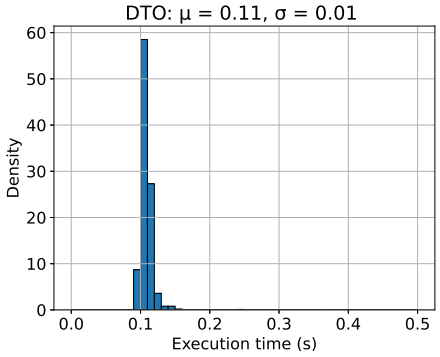
Figure 5.24: The distribution of the execution time when both the server and the client in the raspberry pi 4 device.

5.4.4 Results

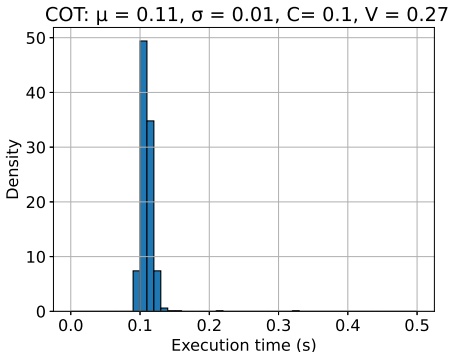
The probability distributions, the average execution times and the standard deviations for the BCP, DTO, COT, Odds, Random and the First option are shown in Figure 5.25. It can be seen that the DTO, COT and Odds models are all Optimal and pick the best server (MacBook Pro 1) as the generated thresholds for these models are in the range of [0.2,0.3]. Therefore the MacBook Pro 1, which has an execution time less than these thresholds is selected for offloading each time. The next best model in terms of expected execution time is the BCP model.



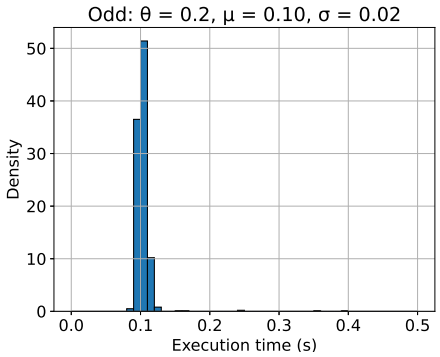
(a) BCP



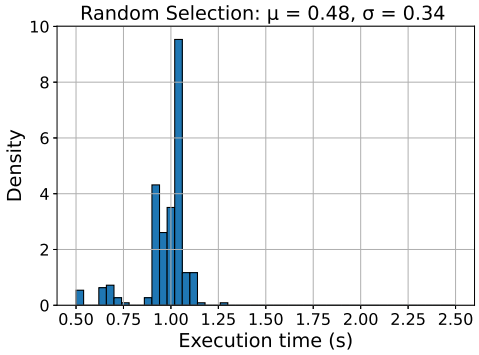
(b) DTO.



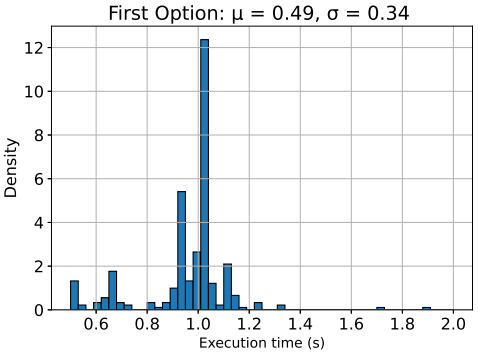
(c) COT.



(d) Odds.



(e) Random Selection.



(f) First Selection.

Figure 5.25: The distribution of the execution times for all the models.

5.5 Mobility Scenarios

As the mobility of the mobile node plays a key role in task offloading within the MEC environment, it is important to consider its effects when applying the OST-based decision-making. Therefore, we simulate a mobile node (smart vehicle) that moves in one direction with different speed values uniformly distributed in $[1, 5]$ meters per second and it passes by a set of MEC servers. The communication range of the MEC servers is 100 meters. Five MEC servers over a distance of 1000 meters were deployed, i.e., one server each 200 meters.

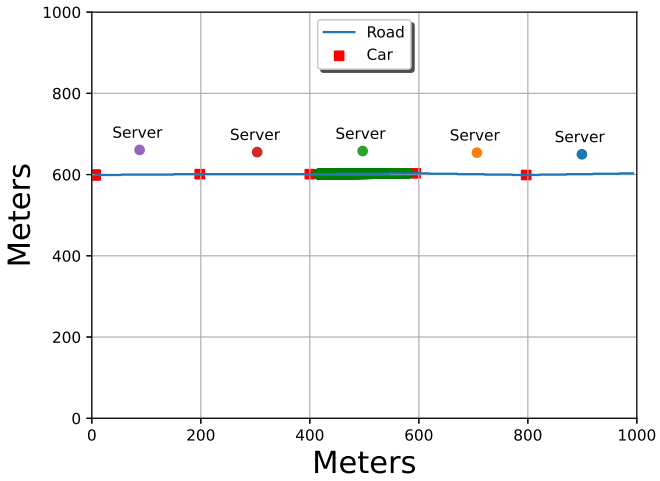


Figure 5.26: Mobility simulation.

Figure 5.26 shows such a setting. As it can be seen from the Figure, the car selects server 3 for offloading. The green line shows that the car was within the communication range of server number 3. Now we define the **traveling time** T to be the time it takes the car from the first point of the green line till the last point of the green line. The car offloads data to be processed before going out of the range. Having such settings, the mobile node will face one situation from two.

First, when we have a processing time less than the traveling time. This case can arise when the load of the MEC server is light, e.g., due to the density of the vehicles being low [5] or the speed of the mobile node is not high. In this case, the OST models have higher probability to select a MEC server that finishes the task before the mobile node gets out of the range of that MEC server with minimised processing time. To check this, a speed $\in [1, 5]$ that generates traveling time higher than the processing time has been used. The processing time are assumed to follow normal distribution, i.e., $X\ \mathcal{N}(50s, 10s)$. The results show that the difference between the processing time and the traveling time when applying the OST models is higher than the other models: the Random and the first selection as shown in Figure 5.27. The higher the difference the more reliability the model has. This indicates that the OST based offloading is more reliable offloading than the other offloading methods as the proposed models ensure the task finishes before going out of the range of the communication.

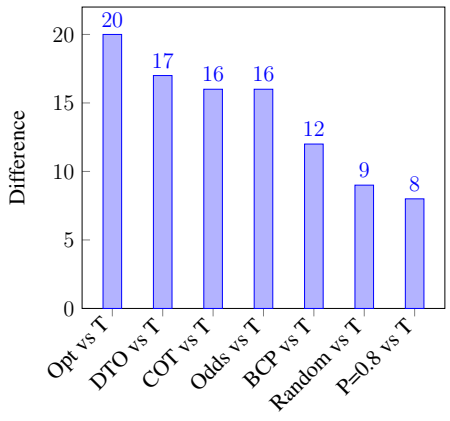


Figure 5.27: Absolute difference between the processing time and the traveling time T.

Second, when the processing time is higher than the time the mobile node spends within the communication range, i.e., $X > T$. In this case, we need a mobility management algorithm that handles such a mobility. Examples of such algorithms are power control for low mobility, and path selection or task migration for high mobility as stated in [13]. The adoption of the OST based selection in such scenario can be benefit. For example, in each method of the mobility algorithms, the first step is to make a selection for an Edge server to offload to. Therefore, it is not difficult to consider the proposed models for this kind of scenario, but this is left for future work and out of the scope of this thesis.

5.6 Discussion

5.6.1 Deployment Environment

The proposed models can be integrated with the current offloading architectures and frameworks. For example, considering the existing work in offloading decision framework in the smartphones devices, in general, the main components for the offloading framework are decision engine or code offloader, network, Edge servers and application profilers, e.g., [14, 25, 63, 66, 70]. The previous components are envisaged to be implemented on a middleware on top of the smart-devices operating system to perform code offloading decision [70]. The offloading engine, in general, is fed with the current state information (e.g., network's state and servers' load) collected by profilers. In the case of smart vehicles, for example, beaconing and resource discovery phase where beacon messages between the smart vehicle and RSUs is used to obtain the required information for the decision-making as presented in [7]. Based on the collected information, the decision engine is expected to give a decision of whether the task should be offloaded or run locally on the mobile device. Our models can

utilise such information and can be implemented on top of existing decision engine components, and it can be triggered whenever the output of the decision is to offload to an Edge server.

5.6.2 Computational Complexity

Regarding the time and space complexity of our models, in general, time complexity in the worst case is $\mathcal{O}(n)$. The mobile node is going to observe n servers if the condition for each model is met at the server number n. For the models DTO, COT and the Odds, there is one step before the observation which is the generation of the thresholds, i.e. $\{a_k\}_{k=1}^{n}$, V^* and s. We assume that this step is to be done once by the services provider outside the mobile node, but it is also not difficult to implement such a step in the mobile node. For example, in the DTO model, we need $\mathcal{O}(n)$ if we calculate the threshold in the mobile node. This is also true for the Odds model. We require more time for calculating the threshold for the COT model, but this depends in the probability distribution. For example, in the uniform distribution, we only perform one operation as shown in Section 4.4. In the normal probability distribution, we calculate and estimate the integration with time complexity no more than $\mathcal{O}(n^2)$. For the space complexity, in the BCP model, we do not need extra space to store any data, thus, the space complexity is $\mathcal{O}(1)$. This is also true for the rest of the models if we assume that the training phase is done outside of the mobile node. In the case where we do the training phase locally in the mobile node, we only need to store the parameters of the probability distribution. When X is uniformly distributed, we need to store the maximum and the minimum values. When X is normally or exponentially distributed, we need the mean and the standard deviation.

5.6.3 Local & Autonomous Decision-Making

It is important to mention that we can achieve full independence for the models DTO, COT and the Odds. For example, in most of the existing offloading framework, e.g., [70] [63] [66], the decision engine is supported by a database where information about the offloading history can be stored. The challenge now is how to estimate the probability functions based on the stored data so we can apply the proposed models. Such estimation can be done using Kernel Density Estimation (KDE) [111]. KDE is a non-parametric method of estimating the probability density function of a continuous random variable. The term kernel refers to a special type of probability density function with the properties: nonnegative and integrates to one. One can estimate the density function $\hat{f}_X(x)$ of a random variable X in an incremental manner at observation $k+1$ using the following equation:

$$\hat{f}_{k+1}(x) = \frac{k}{k+1}\hat{f}_k(x) + \frac{1}{k+1}K_h(\frac{x - X_{k+1}}{h}). \quad (5.1)$$

where $K_h(\cdot)$ is a kernel function and $h > 0$ is a smoothing parameter called the bandwidth. Gaussian kernel function $K(x) = \frac{1}{\sqrt{2\pi}}e^{-0.5x^2}$ of width $h > 0$, is widely adopted in the KDE. A simple method for choosing the value of h is the Sliverman's rule-of-thumb width estimator which is $h \approx 1.06\sigma t^{-1/5}$ [112]. Once the probability distribution function is estimated, we then calculate the thresholds for each model based on the steps and the equations for each models.

5.6.4 Overall Performance and Application Domain/Use Cases

Table 5.4 gives a summary of the models in terms of their performance and examples for tasks to be offloaded for each model taken from the literature. From the results, we noticed that the more information we provide to the model the better results we get. This is clear in the DTO, the COT and the Odds models. In the BCP, however, the random variable X was higher, but we have to consider that we only feed the model with the number of observations. In other words, this model is fully independent, it is very light to be run in the mobile node and performs better than the Random and the p-mode.

In the BCP and the DTO, we generally aim to minimise the random variable without defining further parameters. In the COT and Odds, on the other hand, there are parameters within the models that can be exploited to characterise the model based on the task to be offloaded. In the COT model, the parameter c can be used to define the demand of the task. As an example, we manage to select a server with less processing time (Section 5.2) and less CPU utilisation (Section 5.3) by adjusting the value of c. Therefore, the COT can be used to manage offloading resource-intensive or delay constrained tasks. Moreover, the Odds model can be utilised in the data analytics task offloading where the mobile node collects and senses data with the aim of offloading them to an Edge server for further analysis.

In general, the proposed models can be utilised in applications where there is a deadline by which the task has to be finished. The deadline can be either a hard deadline, in which the computing task must be completed before the deadline or the results would be useless, or a soft deadline, in which some tasks can be completed after the deadline [30]. For example, the use case involving video stream analysis for computation-intensive and delay tolerant service where mobile nodes can upload their video for further processing as the case in recognition applications, e.g., vehicular license plate recognition, face recognition, and home security surveillance [30, 113]. In this type of applications, the time between submitting the

Model	Performance	Applications
BCP Section 3.3	Better than the p-model and the Random	Delay-Tolerant task offloading [18]
DTO Section 4.3	Near Optimal	Delay-Tolerant task offloading [18]
COT Section 4.4	Near Optimal	Resource-intensive and delay constrained tasks [113]
Odds Section 3.4	Near Optimal	Data analytics task offloading [55, 100, 114]

Table 5.4: Lessons to be drawn from this work.

computing task and the deadline can be used to explore more options in terms of times or servers selection.

Another example is where mobile nodes offload contextual information gathered by various types of sensors installed on the mobile nodes. Sensors in smart vehicles, for example, can collect environmental information such as navigation services, temperature, behaviour detection, and images. Offloading such large amounts of data to Edge servers in order to create accessible, trustworthy, and distributed environments can be beneficial for smart city applications [114]. In this case, the deadline can be based on the data to be offloaded and timeliness for such data. It's also worth mentioning that a situation like this can be used to optimise other random variables like the amount of energy used for task offloading, which is heavily influenced by wireless channel conditions [30].

5.7 Summary

This Chapter has presented the most important characteristics of the developed solutions to optimise the task offloading decision. Three evaluation approaches have been used to show the effectiveness of the proposed models. These are;

- a simulated random variable, drawn from three well-known probability distributions (Section 5.2);
- two real data sets are combined to see how the models deal with a real random variable based on the servers' CPU utilisation (Section 5.3); and
- the models are deployed and implemented in real machines (Section 5.4).

In all of these approaches, the proposed models are compared with the Random and the first server selection. All the models are then compared to the Optimal. As mentioned earlier, the Optimal will not be available to the mobile node in the assessed setting, but is used here to assess the performance of the proposed models.

In the simulation approach, the key point is that the proposed models are, in general, near Optimal in that the distributions of the expected processing times are close to the Optimal

solution. Indeed, the proposed models, including the DTO, COT and Odds, are higher than the Optimal by only 10%. In this experiment, the probability distribution of the random variable is known in advance and the goals is to see how the models perform with the normal, the uniform and the exponential distributions.

In the real data sets, however, after studying the whole data set, it is assumed that the probability distribution of the servers' CPU utilisation is approximately following a normal distribution. With that assumption, the models are still near optimal with only around 25% difference compared to the Optimal.

In the real environment assessment, the OTS-based models including the DTO, COT and Odds are all Optimal in selecting a node for offloading. This is due to the small number of devices, making it easier to have the statistical information for the random variable to be optimised. Such an assessment shows how easy it is to adopt the OTS-based models for deployment in the mobile nodes. Table 5.5 summaries the three experiments and their findings.

Experiment	Random Variable Representation	Main Objective	Key findings
Simulation	$\mathcal{N}(50, 10)$ $\mathcal{U}(0, 1)$ $exp(\frac{1}{50})$	Studying different probability distribution functions	Near-optimal processing time
Server utilisation	CPU utilisation	How OST-based models deal with real-world random variable	Near-optimal CPU selection
Real implementation	Task execution time	Applying the models in real scenario	Optimal task execution

Table 5.5: A summary of the evaluation Chapter for all the experiments.

The Chapter concludes with a discussion on important aspects of the models including computational complexity, decision-making requirements and the models domain. In general, time complexity in the worst case is $\mathcal{O}(n)$. The mobile node is going to observe n servers if the condition for each model is met at the server number n. Full independence for the models can be achieved by estimating the statistical properties of the random variable. Finally, the proposed models can be utilised in applications where there is a deadline by which the task must be completed.

Chapter 6

Conclusions & Future Work

6.1 Overview

This Chapter outlines and concludes this thesis. It starts with Section 6.2 where a summary of the contributions of this thesis is presented. In Section 6.3, the thesis statement is revisited. Section 6.4 presents some future use cases inspired by the work presented in this thesis. Section 6.5 presents a number of directions for future work derived from the limitations and the possible extensions to the current work. Finally, concluding remarks are summarised in Section 6.6.

6.2 Contribution Summary

This thesis has addressed the issue of task offloading decision-making in Mobile Edge Computing (MEC) environments adopting the principles of Optimal Stopping Theory (OST). The focus of recent research has been on the offloading decision relating to whether the mobile node should offload or not in the single MEC server setting. This thesis, however, focuses on the sequential decision-making relating to selecting an appropriate resource (MEC server) or time when offloading a computation task in the emerging MEC settings when multiple servers are available for offloading. The work demonstrates the benefits of utilising the optimality found in the context of the OST by designing autonomous and standalone offloading framework to be implemented in the mobile nodes. It starts by providing a theoretical framework of the adopted OST-based models before covering the application of the models in the MEC environments and finishing with a comprehensive evaluation of the models.

The contributions of this thesis are summarised in the following points:

- **A review of Edge computing development and computation offloading**
  This thesis began by reviewing recent development in Edge computing with specific focus on the concept of MEC and its use cases. In addition, the thesis provides a detailed overview of computation offloading and its rules on the emerging architectures. It illustrates the rise of computation offloading applications in MEC environments and how this motivated the need for effective offloading decision-making.

- **Best-Choice Problem based task offloading rule**
  This thesis proposes a time-optimised OST-based model to maximise the probability of offloading to the Optimal server (Section 3.3). The theoretical background of the adopted model, i.e., the Best Choice Problem (BCP), is first presented. The optimal offloading rule is then defined for different numbers of observations. This model only requires the number of observations to make a selection decision, and it outperforms the Random and the immediate offloading. For example, as seen in Section 5.3, the average server CPU utilisation achieved by the BCP model is 40% higher than the average server CPU utilisation achieved by the Optimal whereas the first selection (P-model) and the Random are higher than the Optimal by more than 70%. While this model does not require any information rather than the number of observations, which can be easily obtained, the existing proposed offloading decision frameworks, which included storage resources for past offloading decisions, can hold the past offloading decision data. This stored data can be used for more advanced models that achieve less processing time when offloading.

- **Quality-aware contextual data offloading rule**
  Focusing on the same objective, i.e., maximising the probability of offloading to the best server, a data-quality aware offloading rule is proposed in Section 3.4. The model is based on the optimality of the Odds model; another OST-based model. However, contrast to the Odds original model, a non-increasing function representing the staleness of the data to be offloaded is integrated into the model. In addition to the data-quality indicator, the Odds-based model takes the probability distribution function and a requirement threshold as inputs. It outputs the number of servers that should be rejected before considering an MEC server for offloading. The probability of offloading to the best server is always at least 0.368, which is achieved by the BCP policy with a very large number of MEC servers. Along with the ability to include a data-quality indicator, the average execution time (Section 5.4) achieved by the Odds was around 0.10 seconds while the average execution time achieved by the BCP was 0.29. This selection is Optimal, i.e., it selects the Optimal node for offloading most of the time.

- **Delay-tolerant task offloading rule**
  Moving to the objective of minimising the processing time at the Edge server, a time-optimised model to minimise the expected processing time when offloading is proposed. This model is a threshold-based model where a threshold based on the distribution of the considered random variable, i.e., the processing time, is generated for each time (observation). Three well-known probability distributions that the processing time may follow in the real world scenarios were considered for the threshold generations process including the normal, the uniform and the exponential distributions. The optimal offloading rules were then obtained and analysed for the three distributions. This model has the lowest difference compared to the Optimal in most of the considered random variables. The average processing time (Section 5.2) achieved by this model was around 42 which is higher than the Optimal by only 10%. In the real data set experiment (Section 5.3), the average server CPU utilisation was around 26 compared to the Optimal which has an average of 21.45. In Section 5.4, the average execution time achieved by this model was 0.11 seconds which is Optimal, i.e., it selects the Optimal node for offloading most of the time.

- **Cost-Based task offloading rule**
  In addition, the thesis proposes a cost based model that minimises the expected processing time when offloading to an Edge server. The model considers a function which consists of the considered random variable (e.g., processing time) and a cost that can be defined based on the application requirements. In this model, instead of having to define the number of observations as we did in the previous models, this model is only fed with a cost and it generates a threshold for each cost. The threshold is then compared with each observation. The mobile node can stop the observation process at any point based on the deadline of the task to be offloaded. Similar to the model in the previous contribution, the uniform, the normal and the exponential distributions were considered and analysed. The optimal offloading rules were then obtained and analysed. This model maintains the same level of optimilty achieved by the DTO model. More specifically, the average processing time (Section 5.2) achieved by this model was around 42 which is higher than the Optimal by only 10%. In the real data set experiment (Section 5.3), the average server CPU utilisation was around 28 compared to the Optimal which has an average of 21.45. In Section 5.4, the average execution time achieved by this model was 0.11 seconds which is Optimal, i.e., it selects the Optimal node for offloading most of the time.

- **Comparative assessment and extensive sensitivity analysis**
  Following the presentation of the OST-based models, the thesis then provides comparative assessment and extensive sensitivity analysis of the proposed models with other

offloading methods using numerical simulation, real data sets and an implementation of the models in real devices. Based on the proposed theoretical framework for each model, the offloading rules were tested under different random variables. The distribution of the selections made by each model along the expected value and the standard deviation were presented. The variables for each model were tested for different values. The results show that the OST-based offloading rules are either near-optimal or reach the optimality in some models. In particular, the expected values of the considered random variables selected by the proposed models were near-Optimal (or equal sometimes) to the Optimal (the minimum values) for each offloading decision.

6.3 Thesis Statement Revisited

In this section, the thesis statement is repeated from Section 1.2, while the remainder of this section indicates how it has been addressed. The thesis statement is restated as follows:

> *This research asserts that, by exploiting the mobility of mobile nodes in MEC environments and the deadline of the task, the decision of where and when to offload can be optimised and can be made independently as a standalone decision-making model. To optimise such a decision, this work presents a lightweight framework using the concept of OST to be deployed in the mobile node in order to have lower processing time compared to the immediate offloading and the Random offloading methods.*

This thesis starts with a description of the need for computation offloading for certain applications in the mobile node. Following this, the role of future networks architecture in the provision of low-latency, context-aware, and personalised applications services for mobile nodes is presented, and the limitations of current research are outlined.

In order to optimise the performance of computation offloading, including the processing time required for tasks or the utilisation of more resources when offloading, the idea of using the task' deadline and mobility to delay the offloading in order to secure preferable resources has been considered and evaluated. After understanding some of the fundamentals of the OST-based models, selecting an offloading time based on such models has been applied and evaluated. Multiple well-known probability distributions with real world applications and implementation have been used in the evaluation.

The idea of exploring more options for offloading when it is available in combination with an intelligent decision-making (OST-based models) results in higher performance. Specifically, the results show that the models are either near-Optimal or Optimal with better performance than both the immediate offloading and the Random selection. This thesis shows that it is

very simple to implement the OST-based models in the mobile node, and it is equally easy to manage offloading decisions by gathering the statistical information about the random variable to be optimised. In particular, as the time complexity of the proposed algorithms in the worst scenario equals the number of observations, which is predicted to be relatively small in real-world use cases, the proposed techniques can be implemented and integrated with the current offloading frameworks independently in the mobile node eliminating the need to implement them in an external node, such as a centralised server or controller.

6.4 Use-Cases

This section presents visionary use cases inspired by the work that has been done in this thesis. In high level, these use cases can be divided into three main categories: the expected need for user-oriented computational offloading decision-making, the application of the OST-based models in the Delay Tolerant Computing (DTC) and the need for Optimal Stopping Theory for decision-making problems in future computing paradigms.

6.4.1 User-Oriented Computational Offloading

This thesis envisions a future where MEC servers are deployed in many places similarly to how Wi-Fi access points are today. Such a vision is motivated by the fact that most of the computing and networking functions will be software-oriented instead of hardware-oriented. In other words, it becomes easier now to deploy services on different places (devices) in the network without having to add new components (hardware) to the existing infrastructure. This vision is derived by the trends on SDN and virtualisation technologies. As an example, one can utilise virtualisation technologies (e.g., VMs or docker) to deploy services on low-cost devices (e.g., home routers) [115]. Moreover, utilising software approaches for providing more resources to end users, as the case in task offloading in Edge computing, has many strengths over the hardware approaches (e.g., enhancing the mobile node with designed on-device application-specific integrated circuits) [116]. Examples of the advantages of using software deployment options are speed, cost and flexibility in the deployment (e.g., using Google Play and Apple App stores can help developer introducing new Edge applications rapidly with low cost) [116].

Furthermore, as observed in Chapter 2 that the vast of the task offloading decision studies assume a central control which is in charge of optimising the task offloading and resource management decisions. Such assumption implies a single operator scenarios, i.e., the network and computing resources are managed and owned by one provider. However, large-scale service deployments would necessitate the use of several providers' computing and

network resources [117]. In such a scenario, it is expected that heterogeneous computing and network resources will be pervasive.

Meanwhile, the different forms and usages of mobile nodes are increasing rapidly. Such nodes are required to manage highly advanced applications with a large volume of data collected from a range of sensors and stored in resource-constrained mobile nodes. As a result, the computation offloading roles increase in importance and it is anticipated that the mobile node will become dependent on such a method for task processing and power consumption issues. For example, the main issue that is noticeable by smartphones and wearable devices users today is the power consumed by such devices.

Therefore, the computation offloading for mobile devices should be implemented in the mobile node and the decision-making should be done in an independent manner. For example, when the batteries of mobile nodes are low, it should be possible to opt to offload computing task though the mobile app without relying on other nodes.

In view of the independence offered by the OST-based approaches, this thesis is envisioned to provide a generic framework for such an implementation. As mobile node movements are predicted and normally follow specific patterns, the statistical information fed to the OST-based models becomes visible and easier to obtain. For example, service providers can train OST models using an area's spatial and temporal characteristics and offer these models to mobile nodes as a service (executed in the mobile node) to assist in the task offloading decision-making. Also, developers of Edge applications can also use such models while creating their applications if task offloading is required.

Moreover, as discussed elsewhere in this thesis, the time of execution and the space required by the proposed model in the worst case linearly depend on the number of observations. As the number of observations is not expected to be very large number, the models do not require excessive amounts of resources to run, which renders them smart solutions for user-oriented computational offloading decision-making.

6.4.2 Delay Tolerant Computing

DTC [118] is a concept that refers to the need for agile customised support for applications that can tolerate large delays, yet are important enough to be completed. The argument behind such a paradigm is that other factors such as cost, privacy, and network load, will guide where computational tasks should be offloaded. The tolerant delay given for such applications provides opportunities to choose where such tasks should be offloaded with the aim of improving other dimensions apart from the delay and the power consumption. The term 'computational reliability' has been used in the DTC proposal to refer to other types of parameters that the resource-constrained users might want to optimise, e.g., stability, privacy,

availability, computational capacity and cost [118]. For example, the cost may refer to the task execution, bandwidth, and privacy represents the extent to which data that is being transmitted is sensitive. Figure 6.1 shows the overall idea of the DTC as presented in [118]. It shows the location options of where offloading can occur, along with the expected DTC applications according to the above-mentioned parameters.

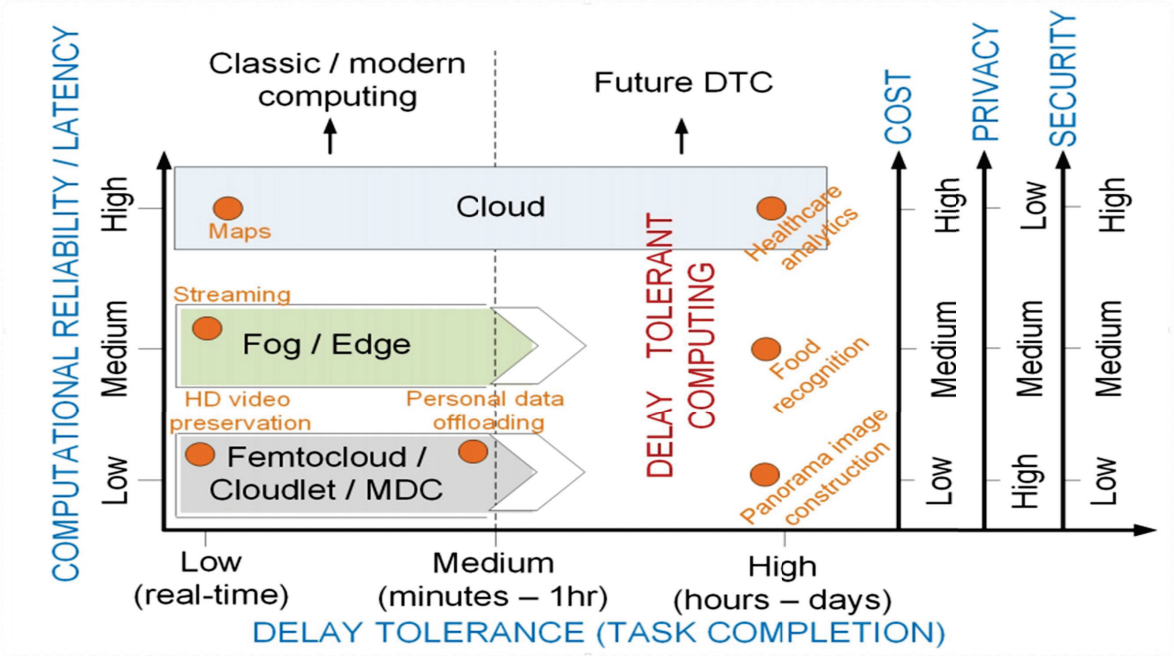


Figure 6.1: The envisioned DTC [118].

As the aims of DTC is to match delay tolerant applications to a wider spectrum of computing opportunities, this work is expected to provide an effective selection of locations where a task should be offloaded based on the envisioned DTC concept. For example, the parameter, computational reliability, can be modelled using OST-based methods, as has been done with different parameters presented in the thesis, e.g., processing time or CPU utilisation. In other words, these parameters can be represented as random variables, which might be following a specific probability distribution. The OST method can then be used to optimise (maximising or minimising) the considered random variable.

6.4.3 OST for Smart Decision-Making

One of the use cases for the MEC paradigm can be used by service providers (network and IT providers) to place their own applications and services on the Edge of the network. Examples of such applications and services include AVs applications, such as those related to safety, convenience, and driving assistance, in addition to applications such as big data and data

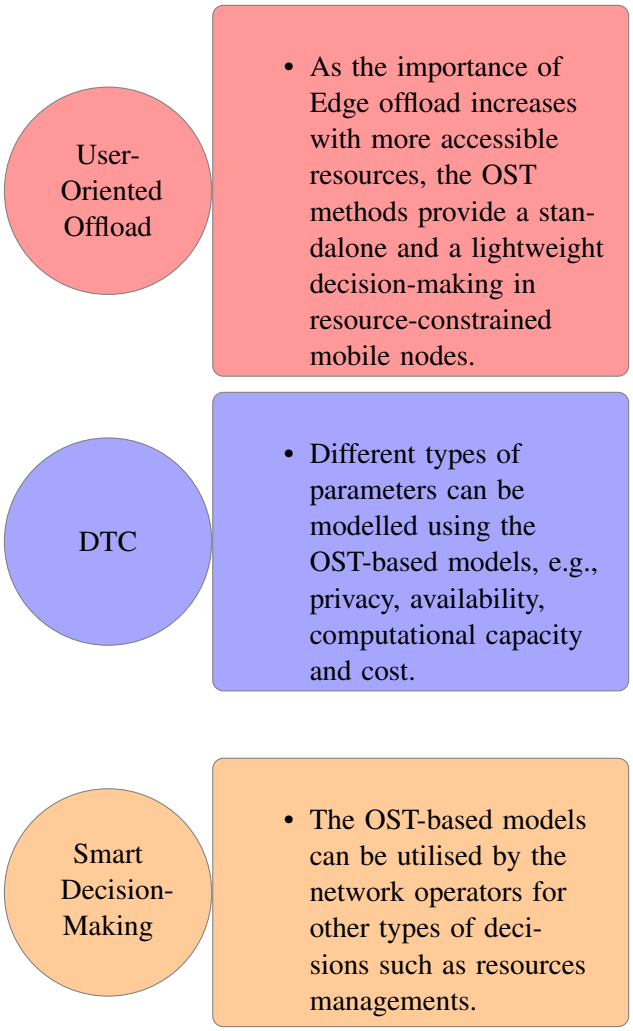


Figure 6.2: Visionary use cases for an OST-based models.

analytics [13]. With the real-time information of such environment in hand, which can also be provided by utilising MEC server [13], service providers need to take many decisions based on this information to optimise the relevant services, as the decision to reshape the traffic per application or to re-route traffic, as required [13].

Through the adaption of the OST demonstrated in this thesis, it appears that such models can be adopted for the purposes of intelligent decision-making for different problems in the MEC environment. This work has considered different statistical information for the computational offloading decision-making. However, the offloading rules and the evaluation provided through the thesis can be extended for application to different problems. Furthermore, there are other forms of the OST-based models (discussed in [22]) that may be suitable for a wide range of problems in such environments. Figure 6.2 summarises possible use cases for an OST-based decision-making system.

6.5 Future Directions of Research

This thesis has highlighted the potential of adopting lightweight OST-based offloading rules in the MEC environment in order to support emerging, resource-intensive mobile node applications. The following paragraphs outline some possible scenarios which could be researched further.

6.5.1 Competitive Scenarios

The models proposed in this thesis work is to be implemented in a single setting, i.e., each mobile node will run the models without taking into consideration other mobile nodes' contexts when offloading. In other words, each mobile node will offload based on the suggestions from the model implemented in that node. As a result, with a large number of mobile nodes, there will be a high probability of different mobile nodes offloading at the same time. Therefore future research should explore the competitive scenario, i.e., what will happen if there is a high probability of the same offloading rules being given to multiple users at the same time. Such a situation could be transferred from a single mobile node to multiple mobile nodes problem. This scenario could also be related to Edge server load balancing.

6.5.2 Different Probability Distribution

Another interesting research direction is the situation where each MEC server has a different probability distribution function. In our experiment, and specifically in the simulation experiment, it is assumed that the processing times for the MEC servers have the same probability distribution functions. However, it could have been interesting to explore whether one can apply the OST based models when there is big difference in terms of the probability distribution functions for the processing time of MEC servers. This might also be related to the probability density estimation method described in Section 5.6. Thus, an interesting research direction would be to explore and experiment this option when there is no information about the probability density function of the random variables to be optimised.

6.5.3 Different Random Variables

This thesis mainly concentrates on the processing time of a computing task in the MEC server. However, the offloading process involves other random variables that need to be optimised. For example, one could study the channel selection and power consumption required for an optimised transmission time and rate based on the data or tasks that need to be offloaded and derived by the probability distribution function of those variables. One

could also combined all the random variables involved in the offloading process. Studies exploring this directions of research should start by analysing the random variable before applying the OST models for optimised selection.

6.5.4 Prediction in Task Offloading Decisions

Another promising research direction is to combine predictive techniques with the proposed models. Predictions can be made in terms of mobility or the load on MEC servers. The proposed models, as discussed throughout the thesis, require some input from the mobile nodes, such as the probability distribution function of the random variable to be optimised. With the user mobility patterns in hand, e.g., the driving habits, one can utilise such information to predict the expected locations of mobile nodes. Combining mobility and load prediction algorithms with the proposed methods can be another source for such inputs. For example, if we have an idea about the expected location of a mobile node along with the expected load of Edge servers deployed in that location, one can obtained the optimal offloading rules in a proactive manner.

6.6 Concluding Remarks

The growth in the number of mobile nodes including smartphones, drones and AVs, and the advances in their applications have brought new challenges to the traditional Cloud architecture. In response to these developments, the Cloud, with its powerful resources, has been offering services to these mobile nodes in order to solve the limitations of these devices. Both academia and industry professionals have proposed several frameworks for moving computing task processing closer to the mobile nodes in order to make the Cloud a more efficient solution. Computation offloading where mobile nodes offload data and tasks to an Edge server is a good way to enhance mobile nodes resources.

This thesis focused on the application of OST approaches in the task offloading decision-making in MEC environments. Derived from the literature, a selection method to be used by the mobile nodes for the MEC servers when offloading is put forward. A detailed assessment of the effectiveness of applying and adopting the OST in the task offloading decision in MEC environments is provided. The experimental evaluation shows that the OST-based models perform better than the other offloading methods, can be used efficiently in the mobile node, and do not require excessive resources.

In particular, the OST-based models are either near-Optimal, with an increase between $40 - 10\%$ over the Optimal, or approach optimality, i.e., the proposed models choose the minimum execution time when implementing them in real machines. Furthermore, the proposed

algorithms have a time complexity of $\mathcal{O}(n)$, where n is the number of observations, and space complexity of $\mathcal{O}(1)$; making them inexpensive to implement by the mobile node.

This work has proved the hypothesis that the offloading decision regarding the selection of an appropriate server can be optimised by exploiting (1) mobility; (2) expected deployment of the MEC server at the Edge of the network; and (3) the deadline of the computational task to delay the offloading in anticipation of finding better resources. This type of decision was formulated as an optimal stopping problem and solved by applying different OST-based models using the optimality of such models in achieving optimal and near optimal offloading resource selection. The OST based model is suitable for situations where the mobile nodes need to make local and independent decisions within the MEC environment. This thesis has shown that we can obtain the required information when making the decision with the aid of the MEC service providers.

Bibliography

[1] G. Brown, "Mobile edge computing use cases and deployment options," *Juniper White Paper*, pp. 1–10, 2016.

[2] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2016, pp. 1–7.

[3] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, pp. 1–28, 2018.

[4] Q.-V. Pham, F. Fang, V. N. Ha, M. Le, Z. Ding, L. B. Le, and W.-J. Hwang, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art," *arXiv preprint arXiv:1906.08452*, 2019.

[5] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[6] W. Tang, X. Zhao, W. Rafique, L. Qi, W. Dou, and Q. Ni, "An offloading method using decentralized p2p-enabled mobile edge servers in edge computing," *Journal of Systems Architecture*, vol. 94, pp. 1–13, 2019.

[7] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling," *IEEE vehicular technology magazine*, vol. 14, no. 1, pp. 28–36, 2018.

[8] D. Sabella, H. Moustafa, P. Kuure, S. Kekki, Z. Zhou, A. Li, C. Thein, E. Fischer, I. Vukovic, J. Cardillo *et al.*, "Toward fully connected vehicles: Edge computing for advanced automotive communications," *5GAA Automotive Association White Paper*, 2017.

[9] R. A. Dziyauddin, D. Niyato, N. C. Luong, M. A. M. Izhar, M. Hadhari, and S. Daud, "Computation offloading and content caching delivery in vehicular edge computing: A survey," *arXiv preprint arXiv:1912.07803*, 2019.

[10] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, 2017.

[11] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[12] C. N. Le Tan, C. Klein, and E. Elmroth, "Location-aware load prediction in edge data centers," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2017, pp. 25–31.

[13] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[14] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, p. 102781, 2020.

[15] Z. Zhou, H. Yu, C. Xu, Z. Chang, S. Mumtaz, and J. Rodriguez, "Begin: Big data enabled energy-efficient vehicular edge computing," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 82–89, 2018.

[16] T. Ouyang, X. Chen, L. Zeng, and Z. Zhou, "Cost-aware edge resource probing for infrastructure-free edge computing: From optimal stopping to layered learning," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 380–391.

[17] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[18] M. Li, P. Si, and Y. Zhang, "Delay-tolerant data traffic to software-defined vehicular networks with mobile edge computing in smart city," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9073–9086, 2018.

[19] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[20] S. Zhou, Y. Sun, Z. Jiang, and Z. Niu, "Exploiting moving intelligence: Delay-optimized computation offloading in vehicular fog networks," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 49–55, 2019.

[21] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.

[22] T. Ferguson. (2020) Optimal Stopping and Applications. [Online]. Available: http://www.math.ucla.edu/~tom/Stopping/Contents.html

[23] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, 2002, pp. 87–92.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[25] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[26] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, p. 107496, 2020.

[27] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[28] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[29] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[30] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[31] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto *et al.*, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2018.

[32] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, pp. 1–24, 2020.

[33] H. Yu, C. Liu, Y. Ren, N. Ji, and C. Yang, "Service node selection optimization for mobile crowd sensing in a road network environment," *Vehicular Communications*, vol. 22, p. 100203, 2020.

[34] W. Liu, Y. Watanabe, and Y. Shoji, "Vehicle-assisted data delivery in smart city: A deep learning approach," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13 849–13 860, 2020.

[35] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[36] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, and M. S. Hossain, "Intelligent task prediction and computation offloading based on mobile-edge cloud computing," *Future Generation Computer Systems*, vol. 102, pp. 925–931, 2020.

[37] A. Shakarami, A. Shahidinejad, and M. Ghobaei-Arani, "An autonomous computation offloading strategy in mobile edge computing: A deep learning-based hybrid approach," *Journal of Network and Computer Applications*, p. 102974.

[38] M. Othman, S. A. Madani, S. U. Khan *et al.*, "A survey of mobile cloud computing application models," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 393–413, 2013.

[39] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[40] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[41] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[42] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[43] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[44] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854–864, 2014.

[45] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Transactions on Mobile Computing*, 2020.

[46] M. ETSI, "Mobile edge computing (mec); framework and reference architecture," *ETSI, DGS MEC*, vol. 3, 2016.

[47] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2016, pp. 1–7.

[48] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, 2017.

[49] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 14–19, 2018.

[50] N. Bhushan, J. Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. T. Sukhavasi, C. Patel, and S. Geirhofer, "Network densification: the dominant theme for wireless evolution into 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 82–89, 2014.

[51] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylinattila, "A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications and technical aspects," *IEEE Communications Surveys & Tutorials*, 2021.

[52] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014, pp. 48–55.

[53] H. Du, S. Leng, F. Wu, and L. Zhou, "A communication scheme for delay sensitive perception tasks of autonomous vehicles," in *2020 IEEE 20th International Conference on Communication Technology (ICCT)*. IEEE, 2020, pp. 687–691.

[54] M. Habib ur Rehman, P. Jayaraman, S. Malik, A. Khan, M. Medhat Gaber *et al.*, "Rededge: A novel architecture for big data processing in mobile edge computing environments," *Journal of Sensor and Actuator Networks*, vol. 6, no. 3, p. 17, 2017.

[55] M. Marjanović, A. Antonić, and I. P. Žarko, "Edge computing architecture for mobile crowdsensing," *IEEE Access*, vol. 6, pp. 10 662–10 674, 2018.

[56] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4568–4578, 2018.

[57] M. H. ur Rehman, C. Sun, T. Y. Wah, A. Iqbal, and P. P. Jayaraman, "Opportunistic computation offloading in mobile edge cloud computing environments," in *Mobile Data Management (MDM), 2016 17th IEEE International Conference on*, vol. 1. IEEE, 2016, pp. 208–213.

[58] A. Alioua, H.-e. Djeghri, M. E. T. Cherif, S.-M. Senouci, and H. Sedjelmaci, "Uavs for traffic monitoring: A sequential game-based computation offloading/sharing approach," *Computer Networks*, vol. 177, p. 107273, 2020.

[59] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[60] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[61] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[62] S. Zhu, L. Gui, J. Chen, Q. Zhang, and N. Zhang, "Cooperative computation offloading for uavs: A joint radio and computing resource allocation approach," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 74–79.

[63] W. Junior, E. Oliveira, A. Santos, and K. Dias, "A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment," *Future Generation Computer Systems*, vol. 90, pp. 503–520, 2019.

[64] A. Ashok, P. Steenkiste, and F. Bai, "Vehicular cloud computing through dynamic computation offloading," *Computer Communications*, vol. 120, pp. 125–137, 2018.

[65] D. Sulaiman and A. Barker, "Mamoc:multisite adaptive offloading framework for mobile cloud applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 17–24.

[66] D. Sulaiman and A. Barker, "Mamoc-android: Multisite adaptive computation offloading for android applications," in *2019 7th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2019, pp. 68–75.

[67] B. Gu, Y. Chen, H. Liao, Z. Zhou, and D. Zhang, "A distributed and context-aware task assignment mechanism for collaborative mobile edge computing," *Sensors*, vol. 18, no. 8, p. 2423, 2018.

[68] S. Xiao, C. Liu, K. Li, and K. Li, "System delay optimization for mobile edge computing," *Future Generation Computer Systems*, vol. 109, pp. 17–28, 2020.

[69] H. Ko, J. Lee, and S. Pack, "Spatial and temporal computation offloading decision algorithm in edge cloud-enabled heterogeneous networks," *IEEE Access*, vol. 6, pp. 18 920–18 932, 2018.

[70] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for iot applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[71] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[72] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[73] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Information Sciences*, 2020.

[74] Y. Ge, Y. Zhang, Q. Qiu, and Y.-H. Lu, "A game theoretic resource allocation for overall energy minimization in mobile cloud computing system," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, 2012, pp. 279–284.

[75] B. Silva, W. Junior, and K. L. Dias, "Network and cloudlet selection for computation offloading on a software-defined edge architecture," in *International Conference on Green, Pervasive, and Cloud Computing*. Springer, 2019, pp. 147–161.

[76] P. Mach and Z. Becvar, "Cloud-aware power control for cloud-enabled small cells," in *Globecom Workshops (GC Wkshps), 2014*. IEEE, 2014, pp. 1038–1043.

[77] P. Mach and Z. Becvar, "Cloud-aware power control for real-time application offloading in mobile edge computing," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 5, pp. 648–661, 2016.

[78] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 2013, pp. 1291–1296.

[79] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1350–1354.

[80] X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[81] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 835–840.

[82] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking Conference (IFIP Networking), 2015*. IEEE, 2015, pp. 1–9.

[83] A. Nadembega, A. S. Hafid, and R. Brisebois, "Mobility prediction model-based service migration procedure for follow me cloud to support qos and qoe," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[84] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2017.

[85] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[86] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive vm handoff across cloudlets," *Technical Report CMU-CS-15–113, CMU School of Computer Science*, 2015.

[87] S. Secci, P. Raad, and P. Gallard, "Linking virtual machine mobility to user mobility," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 927–940, 2016.

[88] Z. Becvar, J. Plachy, and P. Mach, "Path selection using handover in mobile networks with cloud-enabled small cells," in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*. IEEE, 2014, pp. 1480–1485.

[89] J. Plachy, Z. Becvar, and P. Mach, "Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network," *Computer Networks*, vol. 108, pp. 357–370, 2016.

[90] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[91] B. Christian and T. Griffiths, *Algorithms to live by: The computer science of human decisions*. Macmillan, 2016.

[92] F. T. Bruss, "The art of a right decision: why decision makers may want to know the odds-algorithm," *Newsletter-European Mathematical Society*, vol. 62, pp. 14–15, 2006.

[93] R. Dendievel, "New developments of the odds theorem," *arXiv preprint arXiv:1212.1391*, 2012.

[94] M. Sniedovich, *Dynamic programming*. CRC press, 1991, vol. 297.

[95] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the internet of vehicles," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 246–261, 2020.

[96] T. S. Ferguson *et al.*, "Who solved the secretary problem?" *Statistical science*, vol. 4, no. 3, pp. 282–289, 1989.

[97] N. Harth, C. Anagnostopoulos, and D. Pezaros, "Predictive intelligence to the edge: impact on edge analytics," *Evolving Systems*, vol. 9, no. 2, pp. 95–118, 2018.

[98] N. Harth and C. Anagnostopoulos, "Edge-centric efficient regression analytics," in *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2018, pp. 93–100.

[99] C. Anagnostopoulos and K. Kolomvatsos, "Predictive intelligence to the edge through approximate collaborative context reasoning," *Applied Intelligence*, vol. 48, no. 4, pp. 966–991, 2018.

[100] L. Pu, X. Chen, G. Mao, Q. Xie, and J. Xu, "Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 84–99, 2018.

[101] F. T. Bruss, "Sum the odds to one and stop," *Annals of Probability*, pp. 1384–1391, 2000.

[102] C. Anagnostopoulos, "Time-optimized contextual information forwarding in mobile sensor networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 5, pp. 2317–2332, 2014.

[103] C. Anagnostopoulos and S. Hadjiefthymiades, "Intelligent trajectory classification for improved movement prediction," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 10, pp. 1301–1314, 2014.

[104] T. SimPy, "Simpy: Discrete event simulation for python," *Python package version*, vol. 3, no. 9, p. 7, 2017.

[105] J. V. Joseph, J. Kwak, and G. Iosifidis, "Dynamic computation offloading in mobile-edge-cloud computing systems," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–6.

[106] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, p. 100273, 2020.

[107] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," Downloaded from https://crawdad.org/roma/taxi/20140717, Jul. 2014.

[108] "Alibaba cluster trace program cluster-trace-v2018," Downloaded from https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md, November 2018.

[109] "Functions for sequences," accessed on: Feb.15,2021. [Online]. Available: https://docs.python.org/3/library/random.html#functions-for-sequences

[110] "Train image recognition ai with 5 lines of code," accessed on: Feb.17,2021. [Online]. Available: https://towardsdatascience.com/train-image-recognition-ai-with-5-lines-of-code-8ed0bdd8d9ba

[111] A. Zhou, Z. Cai, and L. Wei, "Density estimation over data stream," 2015.

[112] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC press, 1986, vol. 26.

[113] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty, "A survey on task offloading in multi-access edge computing," *Journal of Systems Architecture*, p. 102225, 2021.

[114] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: architecture, applications, technical issues, and future directions," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[115] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *Ieee infocom 2018-ieee conference on computer communications*. IEEE, 2018, pp. 693–701.

[116] M. Satyanarayanan, N. Beckmann, G. A. Lewis, and B. Lucia, "The role of edge offload for hardware-accelerated mobile devices," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, 2021, pp. 22–29.

[117] B. Sonkoly, J. Czentye, M. Szalay, B. Németh, and L. Toka, "Survey on placement methods in the edge and beyond," *IEEE Communications Surveys & Tutorials*, 2021.

[118] M. Aazam, K. A. Harras, and A. E. Elgazar, "Delay tolerant computing: The untapped potential," in *Proceedings of the 13th Workshop on Challenged Networks*, 2018, pp. 33–38.