ENERGY EFFICIENCY COMPARISON FOR LATENCY-CONSTRAINT MOBILE COMPUTATION OFFLOADING MECHANISMS

BY

FENG LIANG, B.Sc.

A THESIS

SUBMITTED TO THE SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

OF THE UNIVERSITY OF OTTAWA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF COMPUTER SCIENCE (M.C.S.)

© Feng Liang, Ottawa, Canada, 2023

All Rights Reserved

Master of Computer Science (2022) University Of Ottawa

(Computer Science) Ottawa, Ontario, Canada

TITLE:	Energy efficiency comparison for latency-constraint mobile computation offloading mechanisms
AUTHOR:	Feng Liang B.Sc., (Computer Science) Oregon State University
SUPERVISOR:	Dr. Azzedine Boukerche
NUMBER OF PAGES:	xv, 116

To my beloved mother

Peiqing

Abstract

In this thesis, we compare the energy efficiency of various strategies of mobile computation offloading over stochastic transmission channels where the task completion time is subject to a latency constraint. In the proposed methods, finite-state Markov chains are used to model the wireless channels between the mobile devices and the remote servers. We analyze the mechanisms of efficient mobile computation offloading under both soft and hard latency constraints. For the case of soft latency constraint, the task completion could miss the deadline below a specified probability threshold. On the other hand, under a hard deadline constraint, the task execution result must be available at the mobile device before the deadline. In order to make sure the task completes before the hard deadline, the hard deadline constraint approach requires concurrent execution in both local and cloud in specific circumstances.

The closed-form solutions are often obtained using the broad Markov processes. The GE (Gilbert-Elliott) model is a more efficient method for demonstrating how the Markov chain's structure can be used to compute the best offload initiation (Hekmati *et al.*, 2019a). The effectiveness of the algorithms is studied under various deadline constraints and offloading methods. In this thesis, six algorithms are assessed in various scenarios. 1) Single user optimal (Zhang *et al.*, 2013) 2) LARAC (Lagrangian Relaxation Based Aggregated Cost) (Zhang *et al.*, 2014) 3) OnOpt (Online Optimal) algorithm (Hekmati *et al.*, 2019a)

4) Water-Filling With Equilibrium (WF-Equ) 5) Water-Filling With Exponentiation (WF-Exp) (Teymoori *et al.*, 2021) 6) MultiOPT (Multi-Decision Online Optimal). The experiment demonstrates that the studied algorithms perform dramatically different in the same setting. The various types of deadline restrictions also affect how efficiently the algorithms use energy. The experiment also highlights the trade-off between computational complexities and mobile energy savings (Teymoori *et al.*, 2021).

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my supervisor, Prof Azzedine Boukerche, for his dedicated support, encouragement, and valuable guidance throughout this research. It was a great opportunity and honor to work under his supervision. His willingness to offer me so much of his precious time and intellect is the main reason for completing this thesis. I extend my warm thanks to teaching assistant Peyvand Teymoori, for her insightful advice and comments.

I am also very thankful for the professors and teaching assistants in the other courses I took during my graduate study. The knowledge gained from them also gives me a huge help on my thesis.

I am also very thankful for the help and support from my father and mother. They give me many advice on learning the knowledge.

Last but not least, I am particularly grateful to my friends in Ottawa for all the fun and unforgettable moments we had together.

Notations

t_r	release time of the job (time slots)
t_D	hard deadline of the job (time slots)
S_{up}	upload data size (bits)
S_{up_1}	upload data size of the first part (bits)
S_{up_2}	upload data size of the second part (bits)
S_{down}	download data size (bits)
T_L	local execution time duration (time slots)
t_L	local execution start time duration (time slot)
T_D	job completion time duration deadline (time slots)
t_D	job completion time deadline (time slot)
T_{up}	upload data time duration (time slots)
T_{up_1}	upload data time duration for the first part (time slots)
T_{up_2}	upload data time duration for the second part (time slots)
T_{exec}	remote execution time (time slots)
T_{down}	download data time duration (time slots)
T_{off}	total offloading time duration (time slots)

t_o	start uploading time (time slot)
t_{o_1}	start uploading time of the first part (time slot)
t_{o_2}	start uploading time of the second part (time slot)
t_f	finish uploading time (time slot)
t_{f_1}	finish uploading time of the first part (time slot)
t_{f_2}	finish uploading time of the second part (time slot)
T_W	elapsed time between two uploads (time slots)
B_{min}	minimum bitrate
B_{max}	maximum bitrate

Abbreviations

3G	Third Generation
CSMC	Channel State Markov Chain
GPRS	General Packet Radio Services
IDC	International Data Corporation
IP	Integer Programming
IT	Information Technology
IaaS	Infrastructure as a Service
LAN	Local Area Network
LTE	Long Term Evolution
MCC	Mobile Cloud Computing
MultiOpt	Mutli-Decision online Optimum)
OnOpt	Online Optimum
PaaS	Platform as a Service
QoS	Quality of Service

SaaS	Software as a Service
TDAMC	Time-Dilated Absorbing Markov Chain
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
GE	Gilbert-Elliott Model
CGAR	Compound Annual Growth Rate
CLE	Concurrent Local Execution
SMD	Smart Mobile Device

Contents

List of Figures

Chapter 1

Introduction

Nowadays, mobile devices are becoming increasingly popular. According to statistics, there will be 18.22 billion mobile devices worldwide in 2025. As a result, our daily utilization of mobile devices will grow. Additionally, an increasing number of devices will be equipped with high-speed connection technology (Statista, 2021).

Consequently, more mobile applications, such as massively multiplayer online games, artificial intelligence, and facial recognition, have been produced (Zhang *et al.*, 2017b). However, computing power and resources are constrained for most battery-operated mobile devices, such as smartphones, tablets, and notebooks. An application with high computing complexity running on a mobile device consumes a large amount of energy, keeps the user waiting for a long period of time, or requires to disable some other functionalities (Yi *et al.*, 2019). Although chips will become more energy efficient as technology advances, mobile applications advance even faster and becomes more energy demanding. Therefore, we also need to research on alternative methods to improve the energy efficiency of mobile devices.

According to a report from Grand View Research, Inc., the market for 5G networks is

expected to reach trillions of dollars by 2030 due to the development of high-speed and low-latency mobile network technologies like 5G (Futurenetworks.ieee.org, 2022) and Wi-Fi 6. The demand for reliable and low-latency networks will grow very quickly, supporting the market growth of 5G. Between 2021 and 2028, the compound annual growth rate (CGAR) is predicted to grow by 50% (Grand View Research, 2022).

The spread of COVID-19, which started in late 2019, also contributes to the increase in mobile device usage. Numerous large corporations, including Google (SHRM, 2021), allow employees to work from home to comply with the social distancing policy. The pandemic catalyzed the demand for mobile devices to rise quickly in the recent years. As more users switch to the 5G network, the number of 5G users will peak at roughly 5.1 billion. The deployment of edge computing hardware at the network's edge and the provision of services to users will increase significantly in the next few years. These advancements can provide the resources for a computationally demanding or time sensitive task focused on mobile devices (Patel *et al.*, 2014). Offloading an application to the cloud is now conceivable due to the network's significantly improved performance. Even the early mobile network had limited bandwidth, early experiments have shown that the mobile cloud computing (MCC) can still provide various advantages, such as extending the battery life of the mobile devices (Rudenko *et al.*, 1998). The mobile cloud offloading (MCO), which is the foundation of the MCC, can increase the mobile device's capacity by offloading a job to a distant server.

The mobile device can get around the MCO's resource and energy constraints by offloading the mobile application to the cloud (Cuervo *et al.*, 2010).

Wireless networks are less stable than wired networks made of fibre and copper cable. The stochastic changes in the wireless channel state affect the network's latency and

speed. Therefore, the offloading decision algorithms must manage these quick changes on the wireless network and ensure that the mobile applications are highly reliable, energy-efficient, and available. Furthermore, either running locally or in the cloud, the application is time-constrained to guarantee the best possible user experience. The variety of wireless channel conditions also adds to the complexity of the offloading decision problem (Zhang *et al.*, 2013). The strict timeline limitation must always be met. Unpredictable channel conditions could result in an outage in unfavorable conditions.

Various single-user offloading decision algorithms have been proposed. In this thesis, we will compare these algorithms by studying their performances in different scenarios. We will evaluate how well the algorithms perform dealing with varying channel conditions, job sizes, and deadline restrictions.

1.1 Mobile Cloud Computing And The Mobile Cloud Offloading

People now use cloud computing in many scenarios of day-to-day life. Many online stores are based on public cloud platforms like AWS. Another type of cloud service that people can access directly is iCloud. Public cloud and private cloud are two separate terms for cloud computing.

The public cloud is for the general public. Any user willing to pay for a public cloud resource is eligible for using it (Liang, x). The most well-known public cloud is the Amazon Web Service. Within a few minutes of setup, users can access various resources on the cloud. The cloud service is more elastic and has a shorter lead time than the traditional data centers.

The advantages of cloud computing for businesses include increased business opportunities and reduced cost (Aldahwan and Saleh, 2018), just to name a few. The private cloud is typically only used internally by the company and is not accessible to the public (Liang, x). To execute their applications, several large businesses have created their private cloud. According to N. S. Aldahwan and M. S. Saleh, the ISO divides cloud services into three service classes: IaaS, PaaS and SaaS (Aldahwan and Saleh, 2018).

1.1.1 IaaS (Infrastructure as a Service)

This is the most fundamental cloud service. It is typically offered as virtual hardware for virtual machines. But it can also give the user access to bare-metal hardware if they need to execute a demanding application. Only hardware and network resources are made available for these types of services. The customer must deploy every program component, including setting up the application's runtime environment and deploying the application on their own servers. Although IaaS requires the user to perform the most amount of work among all the three service classes, it is the most versatile service because users can setup most components and utilize a stand-alone server.

1.1.2 PaaS (Platform as a Service)

Compared to IaaS, PaaS is a higher level of cloud service. It offers the runtime environment for majority of the programming languages. To launch the application, the user only needs to deploy their program and make a few platform-specific settings. For instance, Netlify is a type of PaaS. All users have to do is to import their code from GitHub and run the building commands while observing environment variables. It will then assist the users in deploying the application.

1.1.3 SaaS (Software as a Service)

These applications typically do not require the user to consider the application's deployment status. After creating an account with the supplier, the user may utilize it immediately. One of the most used SaaS services is Microsoft Office 365. After creating an account on Microsoft, users can use it with minimal setup. While focusing on developing their software, Some SaaS providers only focus on developing their SaaS platform and deploy their SaaS applications with other IaaS providers.

One branch of cloud computing is Mobile Cloud Computing (MCC), which uses the network to deliver services such as on-demand access. It can also reserve the network bandwidth to ensure that the user receives the information on time (Sharma *et al.*, 2013). The goal of MCC is to use cloud services to address restricted on-demand options of Mobile Cloud Offloading (MCO) for mobile devices, such as using a wireless network to offload computationally heavy programs to a remote server (Farrugia, 2016). The application should automatically offload, so the user does not need to think about how to run it on both the local device and the remote server.

Shahryar S., Toufeeq Ahmad, et al. proposed to classify MCC's solution into two categories: general-purpose mobile cloud computing (GPMCC) and application-specific mobile cloud computing (ASMCC). GMPCC is based on the system level. It is implemented by hosting a cloud-based copy of the system and then offload the execution of the application from user's device to the copy on the cloud. In this case, the application hardly needs to be adjusted at all. The ASMCC is based on the application level, which requires modification of the application to adapt the MCC (Qureshi *et al.*, 2011).

Although many features of mobile devices have been improved in recent years, their computing power is still insufficient and the users won't always feel that the gadgets are

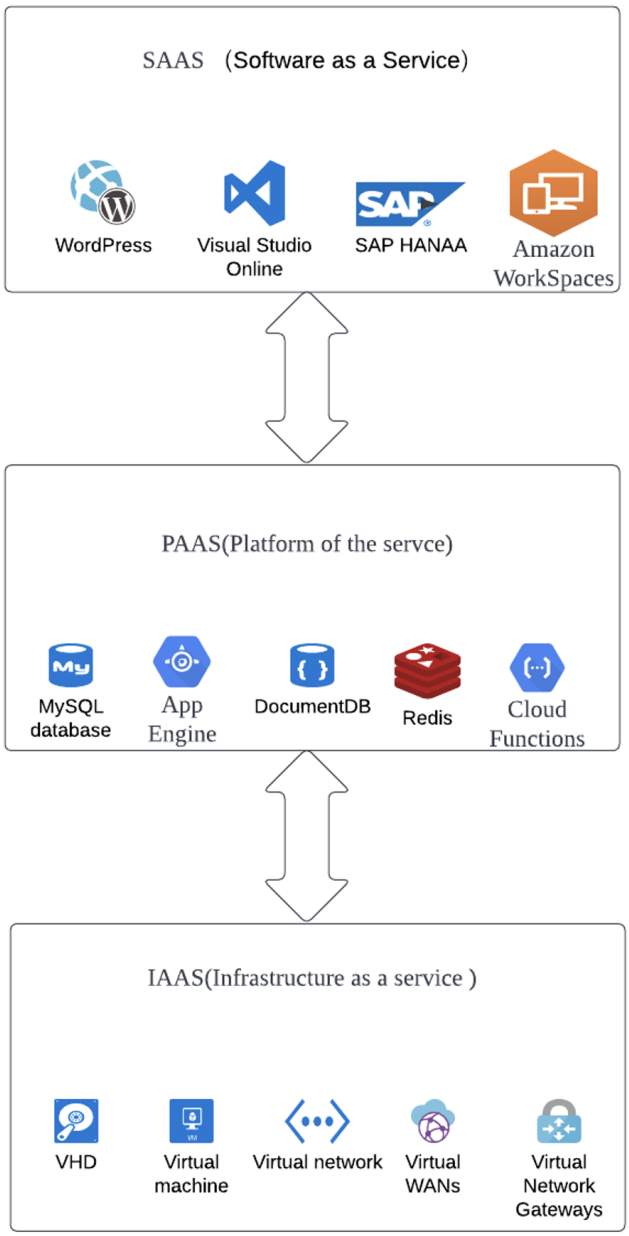

Figure 1.1: The relationship of three kinds of cloud computing service

excessively powerful (Farrugia, 2016).

MCC and MCO are also becoming increasingly popular in the education and health-care sectors, as mobile users seek to replicate the desktop workstation experience while

away from the office. They require a lighter, more energy-efficient solution because the remote desktop based on the VM in the prior solution had extremely high latency (Abolfazli *et al.*, 2014b). The growth of smart mobile devices helps the MCC flourish. However, the popularity of bandwidth-hungry applications like HD online video and cloud storage services is causing an accelerated increase in network traffic, which can also contribute to the excessive latency on the WAN. For the latency-sensitive application, it can be unsatisfactory. Mobile Edge Computing (MEC), a division of MCC that uses a new network architecture to approach MCO, has been proposed. Similar to MCC, MEC is an approach for resource minimization via the MCO. The server will be closer to the end device in the MEC, allowing it to provide higher bandwidth with lower latency (Li *et al.*, 2016). MEC is a technology that will likely take center stage in the future, according to E. Cau et al. By lowering traffic in the core network, it helps reduce network latency and congestion. The development of the higher-level service and support for the new business model is crucial for the next-generation mobile network. Additionally, by moving storage and computing to the edge and structuring a decentralized and distributed infrastructure, the MEC will contribute to enhancing the quality of service (QoS) (Cau *et al.*, 2016).

1.2 Motivation And Contribution

In this thesis, we conduct a comparative analysis of latency-constrained offloading systems that are energy efficient. A proper offloading decision algorithm is required for mobile cloud offloading to decide when and how to offload in a stochastic wireless network environment. The transmission speed, network latency, and bandwidth characteristics frequently vary in a wireless network, making it less stable than a wired network. However, most tasks typically have a latency constraint due to the application scenarios. In addition

to wasting the mobile device's energy, unloading under problematic network conditions will also result in timeouts.

In this article, we explore two different types of deadlines: the soft deadline and the hard deadline. If the latency restriction has a soft deadline, the deadline will be met most of the time rather than always. It is sometimes expressed as a percentage of chance, such as 99% or 99.9%, of meeting the deadline. In contrast, a hard latency constraint must always be met.

Due to the unpredictability of wireless environments, algorithms typically use 'PLAN B' to ensure that the task will be finished before the deadline. When there is just enough time to run the application locally and the result is not received from the remote server, the remote execution and local execution will happen simultaneously.

Numerous offloading decision algorithms based on different deadline constraints have been put forth. Our goal in this thesis is to compare them and analyze the performance of various algorithms across different scenarios to identify their benefits and drawbacks. This can be used to decide the offloading choice algorithms for the MCO depending on the various scenarios of interest. In this thesis, we evaluate the Gilbert-Elliott model performance of six offloading choice algorithms. Additionally, we compare the effectiveness of the offloading choice algorithms under the two often-used deadline constraints, the hard deadline and the soft deadline. Weiwen Zhang, Yonggang Wen, et al. proposed an energy-optimal algorithm, which is the first algorithm (Zhang *et al.*, 2013). This algorithm determines the energy usage for local execution, frequently evaluates the task's ideal CPU state, and then uses a recursive approach to compute the anticipated energy usage for the remote execution. Decisions are based on the anticipated energy for both local and distant execution. The second algorithm was developed by Weiwen Zhang and Yonggang Wen and is known

as LARAC, or collaborative task execution algorithm (Zhang *et al.*, 2014). The work was divided into several smaller tasks, and the challenge was formulated as a shortest path problem on the directed acyclic graph (DAG). The first soft deadline offloading algorithm is a continuing offloading algorithm, and the second soft deadline offloading algorithm is a multipart offloading algorithm. The hard deadline serves as the deadline limitation for the third algorithm. The third algorithm is called OnOpt, and it was developed by A. Hekmati, P. Teymoori, et al. (Hekmati *et al.*, 2019a). It employs a recursive search to determine the ideal offloading start time. However, the recursive time complexity is too high. Thus, the methods are optimized by employing dynamic programming (DP). Preemptive offloading was addressed by developing optimal preemptive offloading (OPO), another online offloading decision technique.

Under the hard-deadline constraint, the OPO algorithms can identify the energy-optimal solution, but the temporal complexity is too high. Therefore, the water-filling technique is used to estimate the outcome of OPO to lower the computing complex of OPO without drastically reducing performance or arriving at a sub-optimal solution. Thus, Peyvand Teymoori, Arvin Hekmati, et al. suggest Water-Filling With Equilibrium (WF-Equ) and Water-Filling With Exponentiation (WF-Exp) (Teymoori *et al.*, 2021). The filling of the water is based technically on 'greedy algorithms.' The WF-Equ method, which uses equilibrium probability to complete the water-filling process, differs the most from the other algorithms studied. The progress of the water filling is made by the WF-Exp employing the transition probability matrix. Under the hard-deadline restriction, the third to fifth methods are both continuing offloading algorithms. The final algorithm is a multipart offloading algorithm called 'MultiOpt' proposed by A. Hekmati, P. Teymoori, and T. D. Todd (Hekmati *et al.*, 2019b). For the OnOpt, it is a multipart version. The k refers to the number of divisions of

the task. If K is set to 1 in this method, the result will be the same as the OnOpt.

Chapter 2 of this thesis will include a review of the relevant literature and related work on the subject. In addition, we will provide some background information on the thesis, such as information on MCC, mobile computing (MC), and MCO. We will summarize the limitations and constraints of the present MCC and MC and compare the benefits and drawbacks of many options of MCC. The task execution on the cloud server and the mobile device will be analyzed, and the issue of migration between the two sides will be discussed. We will talk about the execution model, energy model, and channel model of the thesis in Chapter 3 and explain the execution site and associated energy calculation processes. We will also go over the specifics of the offloading techniques (continuous vs. multipart offloading) and the various hard deadline restrictions. Much more, we will give the problem formulation for the different offloading decision mechanisms. We will detail each of the six algorithms and compare in the experiments in Chapter 4. Chapter 5 will discuss the finite-state Markov chain model and the model reduction methods applied in the experiments. We will also make certain adjustments to various algorithms to guarantee that our trials are more realistic and representative of the real world and then compare the algorithms studied. After this, we will describe the experiments in detail and analyze the results in Chapter 6. Finally, we will conclude the thesis and discuss future works in Chapter 7.

Chapter 2

Literature Review

2.1 Introduction

Overall, the mobile device is playing a more and more important role in daily life, and it will be the most important platform for mobile computing (MC). Cloud computing will also be used to enhance the performance and experience of MC by offloading the computing-intensive application from the mobile device. However, there are many problems with mobile cloud computing (MCC) that still need to be solved. We will review the literature on the different aspects of the MCC, including the topology, limitation, and application model. Some solutions to the problems have also been proposed. Furthermore, we will give a brief review in this chapter.

2.2 Mobile Cloud Computing

Two popular topologies are used to approach the MCC. The centralization topology is the initial topology (Figure 2.1). The centralization topology primarily consist of three

components: a mobile device, a wireless network, and the cloud, all of which are housed in a sizable data center (Yang *et al.*, 2013). There are thousands of servers in the data center, which are connected using high-performance switches (Yang *et al.*, 2013). Because the data center's computational capacity for mobile devices is nearly limitless, the task can usually be completed in a short amount of time, and we need to take the transmission of data into account while offloading.

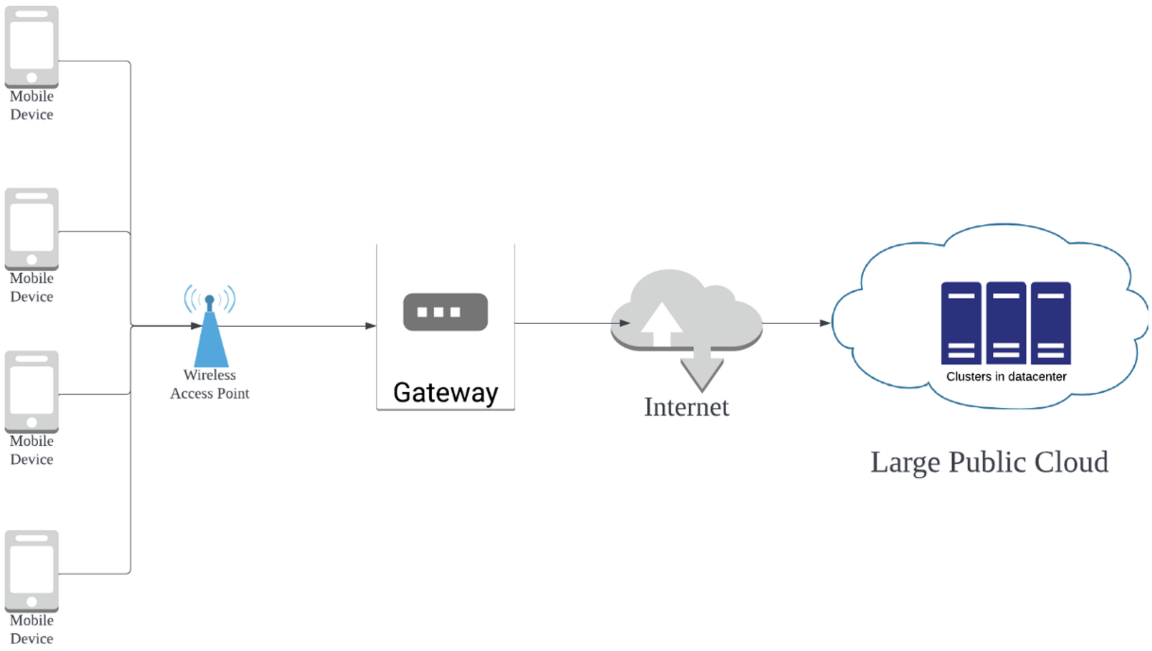

Figure 2.1: The centralization mobile cloud computing topology (Yang *et al.*, 2013)

Although the first topology has several benefits, such as there being no need to worry about computing resource limitations, data must still be transferred over long distances, particularly in locations far from the major data center for the public cloud. Data may need to be transferred, for instance, from one city to another. Because of the higher delay caused by this, the first topology might not be appropriate in situations where a latency-sensitive application needs to be offloaded. Consequently, a different architecture is suggested, one

with a more decentralized structure that deploys the server at the network's edge.

According to Yang et al. (Yang *et al.*, 2019a), it has also been referred to as Mobile Edge Computing (MEC), with the system topology displayed in Figure 2.2. In contrast to the previous topology, the server that will process the work offloading is closer to the mobile device. The infrastructure will be located near the network's edge because the server will be situated so close to the access point. It can assist in reducing ever-increasing network load as well as latency on service delivery (Yang *et al.*, 2019a).

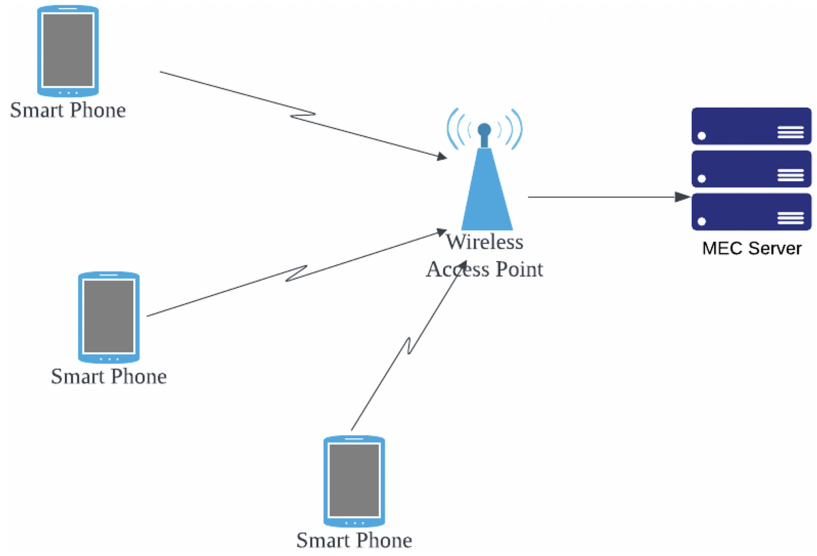

Figure 2.2: The topology on of the MEC (Yang *et al.*, 2019a)

Zhang, Mao, et al. suggested adding backup servers as a method to alleviate the resource limitation on the decentralized topology. A region's base stations seek help from the backup servers when a single base station's servers are too busy to offload all user tasks (Zhang *et al.*, 2017a).

The offloading of the mobile computation to the servers in cloud service is not the only option available to MEC. It is also possible to delegate the task to another user who is prepared to share computing resources with other users in exchange for a reward. The

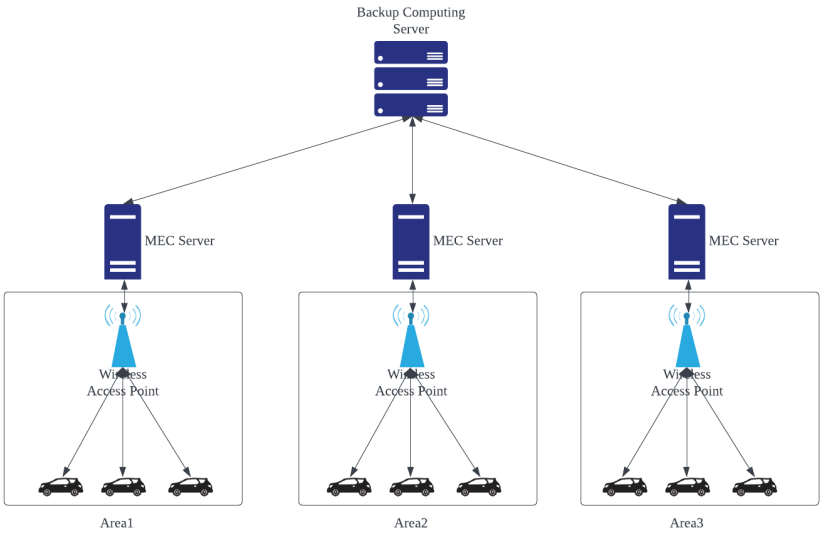

Figure 2.3: The another structure on MEC(Zhang *et al.*, 2017a)

network's processing resources will be more abundant if there are more users, and the system as a whole functions like a scalable distributed system (Habak *et al.*, 2015).

It is also mentioned in the tests concerning the mobile vehicle cloudlets' availability in large-scale urban areas that any device can function in the network as both a client and a service provider, reducing the latency and the cost of transferring the data to the data center in this situation (Wang *et al.*, 2016).

Because of the fundamental system architecture, topology has its own drawbacks. Consequently, a hypertopology has been suggested to improve the topology further. An architecture with five tiers was proposed for the MCC by Yash Chhadva, Ravi Agrawal, and others. Local resources, Ad-Hoc cloud, Cloudlets, private cloud, and public cloud were the five categories into which the offloading was separated (Chhadva *et al.*, 2021). With this architecture, it is possible to transfer the computationally demanding application to a distant cloud. This indicates that it can operate in a sizable cluster without concerns for computing

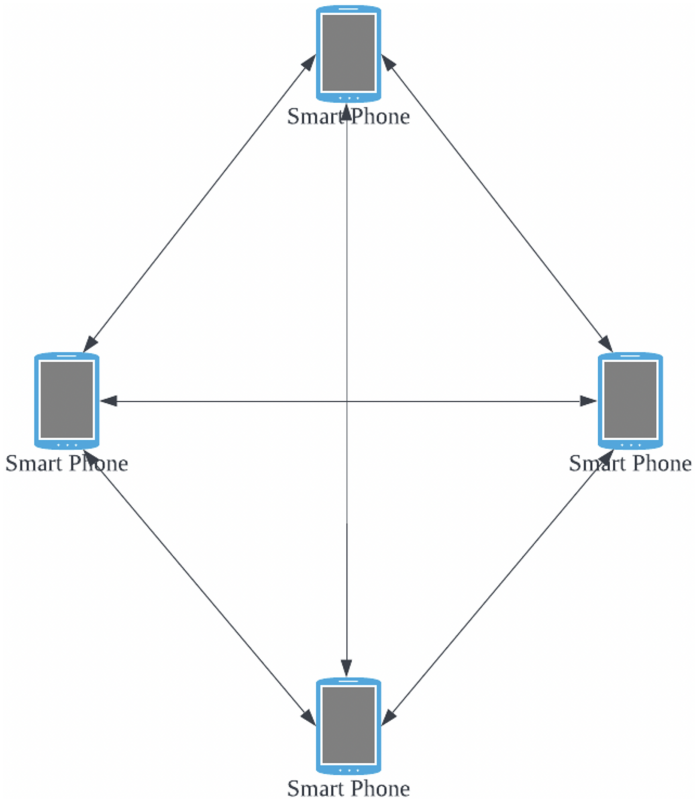

Figure 2.4: The P2P structure of the MCC(Habak *et al.*, 2015)

resource limitations. One could allow processes to operate locally or in the edge cloud if they require a lot of data transfer or are delay-sensitive. This strategy strikes a balance between the amount of computing power available and the network transfer latency.

We must take into account both the application constraint and the topology of the MCC in the real-world setting. E. Cuervo, A. Balasubramanian, et al. have suggested the MAUI, a more practical method that takes the application environment into account (Cuervo *et al.*, 2010). Not everything in the mobile application can be offloaded to the remote server, they said. For instance, some components related to the application's sensor and user interface (UI) cannot be offloaded. For the mobile application, the MAUI will conduct some analysis and identify the methods that can be offloaded using the C# reflection feature.

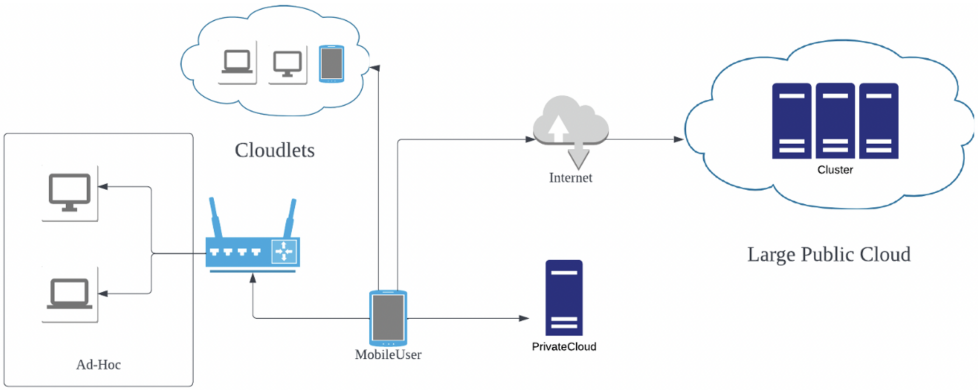

Figure 2.5: The structure of the multi-tier(Chhadva *et al.*, 2021)

Although the MAUI has improved program offloading in real-world settings, it still needs the software developer's assistance. This presents a challenge for some applications because they require more engineering staff to maintain them. Another approach focusing on the JAVA program was created by Y. Zhang, G. Huang, and others (Zhang *et al.*, 2012). This approach is more intelligent since the framework can analyze the application directly and track its progress even when running in the background. The entire application is then refactored in response to demand.

Other frameworks for offloading applications in real-world settings include ThinkAir (Kosta *et al.*, 2012), Cloudaware (Orsini *et al.*, 2016), Cuckoo (Kemp *et al.*, 2010), and CATO (Yang *et al.*, 2017). The application's context should be taken into account when being offloaded.

2.3 The Limitation On The Mobile Computing (MC)

Our daily lives are being influenced by mobile computers. Laptops and other Smart Mobile Devices (SMD), such as smartwatches and smartphones, are the most typical MC devices.

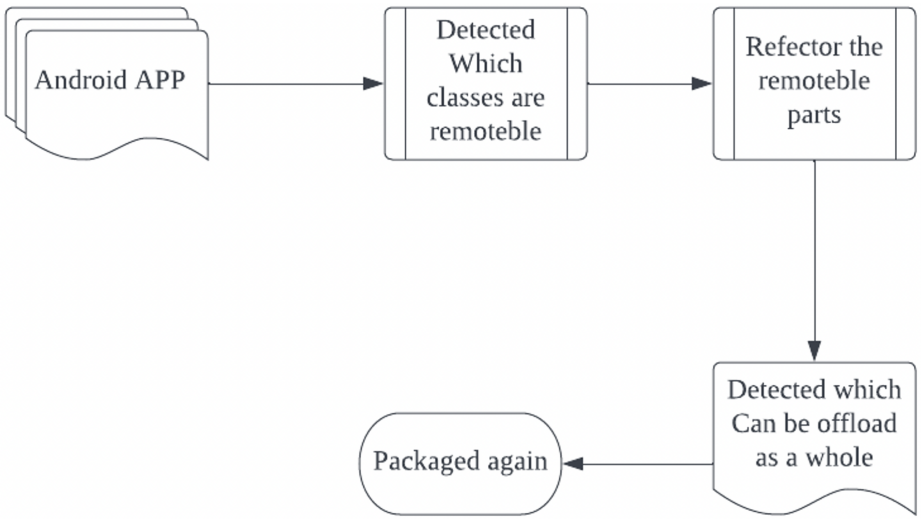

Figure 2.6: A workflow for the reflector of the APP to execute it on remote(Zhang *et al.*, 2012)

They typically include wireless communication characteristics, like cellular or Wi-Fi networks. There are numerous restrictions on mobile computing though (Teymoori, 2022).

2.3.1 Computation Resource

The complexity of mobile applications continues to rise, as does the demand for computational power. The hardware of the devices has improved in terms of computational capability, but it is still insufficient for many complicated applications. Offloading tasks to a remote server, a method known as computation offloading, can be employed to free mobile devices from heavy computations and sophisticated processing (Bangui *et al.*, 2015). This prevents energy-intensive mobile applications from running for a very long time.

2.3.2 Battery Life

Without a power outlet connection, mobile devices rely on their batteries as their main power sources to operate (Forman and Zahorjan, 1994). The battery size is constrained to make the mobile devices smaller, impacting the battery capacity. By lowering battery weight and lengthening battery life, a mobile device's ability to be carried around can be improved.

2.3.3 Memory And Storage

The complexity and range of tasks that can be processed locally are constrained by the memory and storage resource shortage of mobile devices. One of the suggested remedies for this issue is to transfer the data to another device, which is made possible by cloud computing (Mukherjee *et al.*, 2013).

2.4 Network Constraint Of The MCC

The network is one of the most crucial challenges faced by MCC. Throughout the offload, the gadget must constantly transfer the data. Consequently, the MCC requires an ongoing, fast network connection (Qureshi *et al.*, 2011). Even if the cell network is now more robust, in the real world, it is still quite unreliable and unpredictable. The speed and latency of the performance fluctuate erratically. If there are too many users in a certain location, network performance will also suffer (Qureshi *et al.*, 2011). Additionally, it might go down while discharging. It is challenging to meet the strict time constraints with energy efficiency over the random channel conditions. In some cases, concurrent execution on the local and remote environments may be required, which could result in higher energy consumption

than local execution (Teymoori *et al.*, 2021). Under the soft deadline restriction, if the offload decision algorithms choose to offload the work but are unsuccessful in the end, the task must be completed as quickly as possible in local. This will also lead the mobile device to consume more energy. Technology like cloudlets and cached data may be used to mitigate this issue (Qureshi *et al.*, 2011).

2.5 Application Model

The application model describes the system's method of running the program. Before being offloaded, the application typically needs to be partitioned in accordance with certain rules. As previously indicated, some components of the program cannot be offloaded because they require interaction with the device's sensors, such as using the GPS to determine the user's location. Even still, different papers have varying application partition granularities based on things like techniques (Cuervo *et al.*, 2010), components (Kemp *et al.*, 2010), and threads. However, they all analyze the application and determine which portions can be offloaded.

Some methods will also produce a graph showing the relationship between several components, including energy use or other data (Niu *et al.*, 2014). When the analysis is complete, they will decide whether to develop a graph. Focusing on the application context is a different partitioning method because other methods only work in situations where it is almost unnecessary to share the status between the device and proxy (Orsini *et al.*, 2016). By concentrating on the context, the application may be offloaded more effectively and become more scenario-adaptive(Yang *et al.*, 2017).

2.6 Heterogeneous Execution

There are various areas in the MCC where heterogeneity occurs. It may occur on a network or in a distinct program executing environment. Most smartphone SoCs use the big.LITTLE architecture (System On Chip). The 'big' cores are concentrated on high performance, giving our smartphones their best performance. The execution of some computationally expensive tasks, such as playing back high-definition videos and playing games, is typical. The 'little' cores are focused on energy efficiency and low power consumption. They have a micro-architecture that changes regularly. The ideal scenario in a practical setting would be to execute the foreground work using the 'big' core and the background tasks using the 'little' core.

The application context determines the task type (Yang *et al.*, 2017). Furthermore, when work is transferred to the cloud, heterogeneity also occurs. The x86 CPU architecture is typically used in cloud hardware, and the heterogeneity of the hardware expands with the expansion of cloud computing. Although several approaches, such as the use of virtual machines and the unified programming standard, have been proposed, some programs still need to be rewritten in order to work on different CPU architectures. In addition to the heterogeneous hardware, platforms such as Android and iOS and the heterogeneous network must also be taken into account (Sanaei *et al.*, 2013).

2.7 Mobile Access Mechanism

There are several wireless access methods for mobile devices, including LTE, 5G, Wi-Fi, and WiMax, to offload the workload to the remote server or the peers in the network. Its energy usage and performance are different. The many access methods contribute to the

network's heterogeneity as well.

Xing, Yun, et al. have suggested a back-off-based wireless resource scheduling system. Due to the restricted wireless resources, the MCC's service quality varies in a multi-user and multi-service scenario. Therefore, it presents a way to address wireless resource allocation in a multi-user and multi-service environment by prioritizing real-time services to prevent service interruptions and reducing expected global queueing latency (Liu *et al.*, 2016).

2.8 Related Work

Numerous offloading decision algorithms have been developed, and in their work, the authors compare their proposed algorithms to those of other researchers. Other aspects of the MCC have also been the subject of numerous comparative studies. Lung, Srivastava, and Zhan conducted comparison research on the MCC. They contrasted the MCC's energy usage when using various wireless networking techniques and computing platforms. They compared offloading to a remote cloud server through wireless internet access with offloading to a local desktop through Bluetooth (Zhan *et al.*, 2014). Additionally, Ahmadi, Khanezaei, et al. (Ahmadi *et al.*, 2014) conducted a comparative analysis of the MCC's energy consumption and time management across two types of mobile devices and three cloud environments at various distances. They discovered that even if a mobile device is far away from the remote servers, offloading will still help reduce time consumption if the remove servers have higher computational performance. Additionally, shifting the workload to a closer cloud server will help reduce energy usage.

Zhang, Wen, and others proposed energy-optimal techniques for stochastic wireless channels in a publication (Zhang *et al.*, 2013). They demonstrated the architecture of the

cloud-assisted mobile application platform, which involves system-level cloning of every mobile device and virtual machine (VM) operation for the clones. The virtual machine (VM) runs on a server in a nearby cloud infrastructure that can move in response to the user's location. The mobile applications for the platform specified in the article can run either on a mobile device or on a remote server. The goal of the design is to create the best application execution policy while reducing the amount of energy used by the mobile device. By altering the mobile device's CPU clock frequency with Dynamic Voltage Scaling (DVS), the energy computation can be reduced while the application is running on the device. By modifying the data transmission rate in accordance with the channel status, the transmission energy when the program is run on the remote server can be reduced. The two application execution challenges are formulated as convex optimization issues with a soft deadline. They then discovered solutions to convex optimization issues that would result in the mobile device using the least amount of energy (Zhang *et al.*, 2013).

In their article (Zhang *et al.*, 2014), Weiwen Zhang, Yonggang Wen, and others analyzed the problem of how to reduce energy consumption on mobile devices by offloading work to a distant cloud server. They proposed an energy-efficient decision-making procedure for sharing tasks between the remote server and the mobile device. This reference also suggests an enumeration search method and an altered LARAC (LAgrangian Relaxation-based Aggregated Cost) algorithm for locating the best and closest solution for the offloading. According to the article, the one-climb strategy, which calls for only one migration between the mobile device and the cloud clone during execution, is ideal for the linear topology's three-channel models (Zhang *et al.*, 2014).

Concurrent local execution (CLE) is used in the work (Teymoori *et al.*, 2021), which also takes preemptive Mobile Computation Offloading (MCO) into consideration. Under

the preemptive cases, the algorithm determines the solitary upload start time for each time slot. It determines whether to use the current time slot for the upload or postpone it for later time slots based on the supplied channel status and the specifics of the other parameters. When wireless channel conditions change while the computation offload is in progress, preemptive offloading reduces the amount of mobile energy used. These choices are made with the aid of the CLE, such that the system consistently satisfies the specified hard deadline restriction through concurrent local execution offloading. With this restriction in mind, the goal is to reduce the energy usage of mobile devices. In addition, this research developed three computationally efficient strategies based on the online offloading decision algorithm known as optimal preemptive offloading (OPO) (Teymoori *et al.*, 2021).

A decision-offloading method called OnOpt was also proposed by Hekmati, Teymoori, et al. (Hekmati *et al.*, 2019a). This technique theoretically demonstrated that it is possible to obtain the lowest average mobile device energy consumption for homogeneous Markovian wireless channels and satisfies the hard deadline requirements of the applications. Additionally, it uses the CLE to guarantee that the hard-deadline restriction is met. They conduct research on mobile computation offloading in situations where strict time deadlines are imposed on project completion time. Likewise, their work also compared OnOpt with local execution, channel threshold, and immediate offloading. Additionally, these algorithms make sure that strict job deadlines are maintained. A strict lower bound on the energy consumption of mobile devices is calculated using an integer program (IP), which provides a bound for comparisons in their experiments (Hekmati *et al.*, 2019a).

With OnOpt, using the multi-decision online optimum (MultiOpt) algorithms is suggested because their superiority is demonstrated by the best Markovian stopping theory (Hekmati *et al.*, 2019b). The study conducts research on multi-decision offloading for

mobile computation. This happens when a task that needs to be completed remotely is uploaded in sections rather than all at once. When wireless channel circumstances vary while the computation is being offloaded, multi-decision offloading may reduce the energy consumption of mobile devices. The upload is broken up into portions. Thus, the best start time for each portion needs to be chosen. The paper's multi-decision problem is seen as a problem with a hard-deadline restriction. The MultiOpt and OnOpt are identical when the number of sections is set to 1. Additionally, they conducted an experiment and contrast the results with those of four other algorithms, and the findings demonstrate that MultiOpt can enhance the energy efficiency for the mobile device (Hekmati *et al.*, 2019b). According to an evaluation of four algorithms that concentrate on the hard-deadline restriction, even if they use various offloading decision-making processes, all of them make an effort to identify the ideal offloading start time. To ensure that the hard-deadline limitation is met, they all use the CLE.

A model was put forth in the reference (Terefe *et al.*, 2016; Teymoori, 2022) to illustrate how multisite application execution uses energy. The channel environment was modelled in the study using a two-state discrete-time Markov chain (DTMC), which is probably the GE model and uses the status 'Bad' and 'Good.' Using the DTMC, they determined the communication channel's average bitrate of transmission. To make matters worse, they used an MDP framework to construct the multisite partitioning problem as a weighted, time-constrained shortest-path problem on a state transition graph. They suggested a value iteration technique that was the foundation for the Energy-Efficient Multisite Offloading Policy (EMOP) algorithm, built on a value iteration algorithm (VIA). The multisite partitioning problem can be solved using their suggested algorithm. When they solved an optimization problem, they did, however, make some assumptions about the channel state.

Therefore, their suggested approach is unable to handle unstable channel circumstances, and when channel states unexpectedly change, offloading decisions become inefficient and, specifically, less energy efficient.

Yang, Lixin Li, and others (Yang *et al.*, 2019b)suggest that the latency issue with mobile cloud offloading can be optimized by employing the RL technique known as DQN (Deep Q-network). On the deep neural network, the DQN algorithm combined Q-learning, a branch of the reinforced learning approach. The DNN (deep neural network) is used as the Q-value network in their study, and the update problem on the Q-table in the Q-learning is then allowed to devolve into a function-fitting problem, as is typical with most machine learning advancements. Due to a vast state-action space in the Q-table, the algorithms require excessive computing resources to obtain the best answer if they continue to employ the conventional Q-learning techniques. To obtain an approximation of the solution, employing the DQN is appropriate. They also expanded the NOMA-MEC optimization study from a two-user situation to a multi-user situation, which made the NOMA-MEC study more appropriate in real-world scenarios. Multiple users were paired in the study, and the two users partnered well can offload the task concurrently in NOMA mode (Yang *et al.*, 2019b).

Navin J.R. and Lutimath employed k-means clustering in their EAKOS (JR and Lutimath, 2020) method for offloading the Android application to the cloud. They pointed out that since extending battery life has been a significant problem in mobile cloud computing, the MCO can increase the mobile device's performance by minimizing resource consumption. Depending on how frequently a user uses a module, how much CPU time it uses overall, and how much memory it uses, the KNN is used to determine which modules in the program should be offloaded. In addition, they take the wireless network's round time

trip (RTT) into account when conducting the experiments. They used k-means clustering to decide when the device's battery life falls below 50%. The testing results in their study showed that the proposed methods reduce energy consumption by 69% compared to energy consumption of the current offloading schemes.

The reference (Abolfazli *et al.*, 2014a) by Abolfazli, Sanaei, and others mentioned a mathematical model and performed the benchmark on actual mobile devices to analyze the influence of the distance of the cloud server and intermediate hops on the performance of mobile applications when distant cloud resources are used to strengthen it. Their research also shows the viability of utilizing service-oriented computing (SOC) and the RESTful architectural style to improve mobile devices with limited resources. Representational state transfer (REST) is an architectural framework for online services that are used to perform operations on resources. It approaches CRUD on web resources and is more efficient than conventional frameworks. SOC is a computing model for constructing complicated systems out of independent and pre-built processing units with standardized interfaces called services. The analysis examines the advantages of employing the RESTful SOC paradigm in cloud-based augmentation systems and evaluates the impact of intermediary hops on performance. Utilizing the SOC helps to cut down on the overhead associated with code offloading. The research evaluates the time and energy usage of two resource-intensive mobile applications (RMAs) under three different execution policies, including native, proximate cloud, and distant cloud. The experiment's findings showed that leveraging distant cloud to augment mobile devices resulted in substantial unexpected time and energy overheads that reduced application performance and interfered with the correct time and energy estimation. When data communication increases, these effects are amplified (Abolfazli *et al.*, 2014a).

According to the article (Mazza *et al.*, 2014), wireless communication is a key component of smart cities, which are known as a paradigm that improves urban services and raises residents' and tourists' quality of life. The wireless network, various cloud data centers, and a substantial number of access points are just a few of the components that make up the smart city scenario. Many individuals and gadgets exchange a great deal of data in order to provide the needed services, and this data must also be vetted to provide users with accurate information. They also noted that a partial offloading technique could take advantage of the HetNet's situation and the existence of MCC devices by optimizing the quantity of partial offloading of the computational workloads according to their location and the number of devices connected to the network. More importantly, by taking partial offloading into account in a non-trivial multi-objective optimization solution where both energy consumption and delay constraints are solved, their study also resolved the optimization of the complete system, not just on a single device. In their suggested method, a cost function is utilized to assess the trade-off between the energy consumption of mobile devices and the time to offload data and compute operations to a remote cloud server, determining the best offloading percentage based on network load. They also offered a feature that centralized network management can be used to assess the appropriate percentage amount to offload when there is a lot of activity, when the network is congested, and when jobs need a lot of processing and data exchange.

Using a Markov chain, Wu and Wolter suggested a wireless network availability model (Wu and Wolter, 2017; Teymoori, 2022). Two groups can be made via the application:

those can tolerate delays and those can't. The delay-tolerant applications are less sensitive to network delays, such as iCloud, social networking, and mobile healthcare. Delay-sensitive applications include facial recognition, video conferencing, vehicular communications, and authentication because they need a quick response time comparable to ordinary human cognitive capacities. A stochastic offloading technique that supports two separate delayed offloading rules was mentioned in this paper. The first situation is known as a partial offloading model. In this case, the tasks can opt to complete the slow portion of the offloading process locally. The second situation is a full offloading model. The jobs are offloaded straight through the cellular network, bypassing the Wi-Fi queue in this situation. The author also noted that in two of the situations, lowering the energy-response time weighted product (ERTP) metric is the primary goal. Partial offloading is preferred when offloading delay-sensitive applications. When handling a delay-tolerant application with a longer deadline, full offloading is a more effective option. By selecting wireless communication techniques for offloading, the authors constructed queuing models to exploit cellular and Wi-Fi networks. In their experiments, they assumed that connectivity would only be available sometimes and found that if Wi-Fi was scarce, a lot of jobs would give up the wait. The partial offloading model is better suited for applications with medium deadlines that are delay-sensitive. For applications that can tolerate delays, the full offloading strategy works better and can save energy usage. The authors noted that it was possible to determine an ideal deadline to halt offloading progress in both the partial offloading model and the full offloading model in a Wi-Fi setting. However, the assumption is made that the transmission rates of the cellular and Wi-Fi networks are constant in both partial and full offloading regimes. This is not useful in the actual world. Therefore, meeting a challenging job deadline cannot be ensured (Wu and Wolter, 2017).

The study (Li *et al.*, 2022) made the point that the limitation of computational power and battery lifetime prevents the smooth operation of the majority of resource-intensive mobile apps in SMDs. MCC is suggested as a way to lessen the effects of the restrictions. A typical MCC scenario in a heterogeneous wireless network is considered by the author (HetNets). If a cellular network with greater coverage is not available, the device can connect to the network using a short-range connection, such as Wi-Fi. Due to Wi-Fi's generally greater transmission rates and lower network overhead because of its short range and open spectrum resource, mobile devices are more likely to use Wi-Fi if it is accessible. It is difficult to determine the real-time channel status for the Wi-Fi, and it is simple to break a Wi-Fi connection. So in order to enhance the system offloading utility and employ low-cost Wi-Fi, some processes may experience processing delays owing to backlogs in the queue. Consequently, a difficulty arises when deciding how to balance the system offloading utility with the queue backlog. To create online control algorithms that both optimize certain performance goals and stabilize network queues, the study (Li *et al.*, 2022) suggested an effective methodology. Their algorithms also describe a multistage stochastic programming technique used to determine the ideal offloading workloads, which takes into account features for various network types, such as Wi-Fi connections in Wi-Fi-cellular HetNets (Li *et al.*, 2022).

The offloading decision algorithm utilizing greedy was proposed by Li, Huang, and Qian (Li *et al.*, 2021). A new scenario that includes work offloading for vehicular cloud was also mentioned, along with much more. In this case, the deadline restriction and reliability requirement must be met while still maximizing the speed of the offloading methods. Additionally, they formulate their issue as maximizing the submodular function and

demonstrate that it is an NP-complexity task. They investigated this issue online and suggested methods with an O(log2T) time complexity. The authors' proposed algorithm uses a greedy method to determine cardinality. These outcomes are referred to as a location set, and the highest reliable location that is not in the location set is used to replace the least reliable location in the set. They also suggested a relative method that, in the context of random bandit and adversary bandit, provides a sublinear regret guarantee. Their studies reported in the journal (Li *et al.*, 2021) showed that their method outperformed other algorithms in the SUMO environment by 20%.

The mobile data offloading issue is studied in the reference (Liu *et al.*, 2017; Teymoori, 2022) using a mobile cloud computing architecture and stochastic wireless channels. Cellular, Wi-Fi, and device-to-device (D2D) communication networks can all be used to transport mobile data. By utilizing the cellular network less and meeting deadline requirements, they hope to lower the offloading cost. The D2D and Wi-Fi links are used in the article's model to offload some cellular data traffic. The authors use a Markov decision process with a finite horizon to represent the data offloading issue (FHMDP). Although they use a hybrid offloading approach to address the issue for applications that are both delay-sensitive and tolerant, the deadline limitation is not always met. They established a penalty cost function for unsuccessful data transmissions, and by optimizing it, they increased the proportion of offloaded activities as much as possible.

The article (Lei *et al.*, 2015) noted that the problem of the restriction of data storage space and computing power on the mobile DB can be solved by utilizing cloud databases. Software providers began utilizing the public cloud to launch their mobile applications in response to the industry's rapid adoption of cloud computing for data management, storage, and access on mobile devices. This cloud-based approach can offer excellent scalability

for mobile applications and eliminate data size concerns. However, the current approach, which is based on existing cloud databases, has some drawbacks when it comes to supporting a particular ideal mobile database in several areas, such as mobility-based data transactions, simple and seamless integration, and connectivity with mobile application servers. It is not user-friendly enough for users who frequently lose network connectivity. To address these issues, the research has presented a brand-new mobile-enabled cloud database service called MCloudDB. To facilitate mobile data access services with high scalability, it offers a mobile-enabled cloud database service framework that can be used as a bridge to connect and integrate mobile apps with a back-end cloud database.

Table 2.1: The comparison and summary for the literature review

	Title	Contribution
(Cuervo *et al.*, 2010)	Maui: making smartphones last longer with code offload	MAUI is a more practical method that takes the application constraint and the topology of the MCC into account.
(Futurenetworks.ieee.org, 2022)	IEEE Future Networks: Enabling 5G and Beyond	

(Qureshi *et al.*, 2011)	Mobile cloud computing as future for mobile applications-Implementation methods and challenging issues	the MCC requires an ongoing, fast network connection to transfer the data.
(Yang *et al.*, 2013)	A framework for partitioning and execution of data stream applications in mobile cloud computing	The centralization topology primarily consist of three components hosted in a data center with thousands of servers.
(Yang *et al.*, 2019a)	Latency optimization for multi-user NOMA-MEC offloading using reinforcement learning	In Mobile Edge Computing, the server which processes the work offloading is closer to the mobile device, reducing network load and latency on service delivery.

(Zhang *et al.*, 2017a)	Optimal delay constrained offloading for vehicular edge computing networks	Backup servers can be used to alleviate the resource limitation on the decentralized topology when a single base station's servers are too busy.
(Chhadva *et al.*, 2021)	Architecture for Mobile Cloud Computing Using Five Level Offloading (ArMFLORA)	A hypertopology with five tiers for the MCC can improve the topology.
(Zhang *et al.*, 2012)	Refactoring android java code for on-demand computation offloading	The Java program can analyze the application directly and track its progress even when running in the background.
(Habak *et al.*, 2015)	Femto clouds: Leveraging mobile devices to provide cloud service at the edge	The offloading of the mobile computation to devices of other users could make MEC more scalable.

(Wang *et al.*, 2016)	On the serviceability of mobile vehicular cloudlets in a large-scale urban environment	Any device can function in the network as both a client and a service provider, reducing the latency and the cost of transferring the data to the data center.
(Kosta *et al.*, 2012)	Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading	ThinkAir is another framework for offlodaing applications in real-world settings.
(Kemp *et al.*, 2010)	Cuckoo: a computation offloading framework for smartphones	Cuckoo is another framework for offlodaing applications in real-world settings.
(Yang *et al.*, 2017)	Context-aware task offloading for wearable devices	CATO is another framework for offlodaing applications in real-world settings.

(Orsini *et al.*, 2016)	Cloudaware: A context-adaptive middleware for mobile edge and cloud computing applications	Focusing on the application context is a different partitioning method if sharing the status between the device and proxy is necessary.
(Teymoori *et al.*, 2021)	Preemptive Mobile Computation Offloading With Hard Deadlines and Concurrent Local Execution	Concurrent local execution satisfies the specified hard deadline restriction through concurrent local execution offloading.
(Niu *et al.*, 2014)	Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications	A graph showing the relationship between several components such as energy use or other data can be developed.
(Sanaei *et al.*, 2013)	Heterogeneity in mobile cloud computing: taxonomy and open challenges	Heterogeneous platforms such as Android and iOS and the heterogeneous network must be considered.

(Liu *et al.*, 2016)	Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing	A back-off-based wireless resource scheduling system address wireless resource allocation in a multi-user and multi-service environment.
(Zhan *et al.*, 2014)	A comparative evaluation of computation offloading for mobile applications	MCC's energy usage varies for various wireless networking techniques and computing platforms.
(Ahmadi *et al.*, 2014)	A comparative study of time management and energy consumption in mobile cloud computing	Remote servers can help to reduce application execution time despite of the time for offloading.

(Zhang *et al.*, 2014)	Collaborative task execution in mobile cloud computing under a stochastic wireless channel	An enumeration search method and an altered LARAC (LAgrangian Relaxation-based Aggregated Cost) algorithm can be used to locate the best and closest server for the offloading.
(Hekmati *et al.*, 2019a)	Optimal mobile computation offloading with hard deadline constraints	A decision-offloading method called OnOpt theoretically could obtain the lowest average mobile device energy consumption for homogeneous Markovian wireless channels and meet the hard deadline requirements of the applications.

(Hekmati *et al.*, 2019b)	Optimal multi-decision mobile computation offloading with hard task deadlines	With multi-decision online optimum (Multi-Opt) algorithms, a task is uploaded in sections rather than all at once to reduce energy consumption.
(Zhang *et al.*, 2013)	Energy-optimal mobile cloud computing under stochastic wireless channel	Energy-optimal techniques for stochastic wireless channels involves system-level cloning of every mobile device and virtual machine (VM) operation for the clones.
(Yang *et al.*, 2019b)	Latency optimization for multi-user NOMA-MEC offloading using reinforcement learning	The latency issue with mobile cloud offloading can be optimized by employing the RL technique known as DQN (Deep Q-network)

(JR and Lutimath, 2020)	EAKOS-Energy Aware KMeans Offloading Scheme for Mobile Applications	K-means clustering in the EAKOS method can be deployed for offloading the Android application to the cloud.
(Abolfazli *et al.*, 2014a)	An experimental analysis on cloud-based mobile augmentation in mobile cloud computing	A mathematical model is used to analyze the impact of cloud service distance and intermediate hops on the performance of mobile applications. Service-oriented computing (SOC) and the RESTful architectural style can also improve mobile devices with limited resources.
(Mazza *et al.*, 2014)	A partial offloading technique for wireless mobile cloud computing in smart cities	Wireless communication improves urban services and enhances quanlity of life in smart cities.

(Wu and Wolter, 2017)	Stochastic analysis of delayed mobile offloading in heterogeneous networks	In actual world, meeting a challenging job deadline is not ensured becasue transmission rates of the cellular and Wi-Fi networks are constant in both partial and full offloading regimes.
(Li *et al.*, 2022)	Lyapunov optimization based trade-off policy for mobile cloud offloading in heterogeneous wireless networks	The limitation of computational power and battery lifetime prevents the smooth operation of the majority of resource-intensive mobile apps in SMDs
(Teymoori, 2022)	Efficient Mobile Computation Offloading with Hard Task Deadlines and Concurrent Local Execution	There are numerous restrictions on mobile computing.

(Li *et al.*, 2021)	Latency-aware Batch Task Offloading for Vehicular Cloud: Maximizing Submodular Bandit	The offloading decision algorithm utilizing greedy was proposed to determine cardinality.
(Lei *et al.*, 2015)	MCloudDB: A Mobile Cloud Database Service Framework	The problem of the restriction of data storage space and computing power on the mobile DB can be solved by utilizing cloud databases.
(Bangui *et al.*, 2015)	External sources for mobile computing: The state-of-the-art, challenges, and future research	Offloading tasks to a remote server prevents energy-intensive mobile applications from running for a very long time.
(Forman and Zahorjan, 1994)	The challenges of mobile computing	Mobile devices rely on their batteries as their main power sources to operate and the battery size is constrained.

(Mukherjee *et al.*, 2013)	Femtocell based green power consumption methods for mobile network	Transferring the storage to another device can help to solve the memory and storage resource shortage of mobile devices.

Chapter 3

System Model And Problem Formulation

3.1 Introduction

In this chapter, we introduce the different latency constraint mobile computation offloading techniques. We will present the system model, including a task profile, local and remote execution schemes, and a channel model for each technique. We will study the problem of mobile computation offloading for both soft and hard deadline constraints and compare the result in terms of energy consumption and execution time in the subsequent chapter. Under soft deadline constraints, we consider selective and collaborative offloading, respectively. In the selective offloading method, the entire task will be executed with either a mobile device or the remote cloud server; therefore, only one offloading decision should be made at the release time of the job. In a collaborative execution scheme, each workload piece of the task can be executed locally or remotely, thus the offloading decision should be made for each particular workload piece of the task.

Under the hard deadline constraint, we consider continuous offloading, multi-part offloading, and preemptive offloading. In continuous offloading, the entire task has only one offloading decision in the whole progress. With multi-part offloading, the entire task has a given number of offloading decisions. The preemptive method has one or more offloading decisions according to the wireless channel environment.

We use the Markov Chain as the channel model in our thesis. It is a popular method for describing the wireless fading channel (Dehnie, 2007). We use the Glibert-Elliot (GE) channel models and the TDAMC channel model in the paper. The GE model is a special case of the Markov Chain channel model, which only has two states. The TDAMC model combines the time series with the Channel State Markov Chain.

3.2 Mobile Computation Offloading Under A Soft Deadline Constraint

In this section, we will give details about two offloading techniques under the soft deadline constraint. As aforementioned, the soft deadline sometimes remains unsatisfied. We will give the details, including the task model, the energy and time model for local and remote execution, the channel model, and the problem formulation for selective offloading and collaborative offloading.

3.2.1 Selective offloading

This subsection will explain task execution locally and remotely under the soft deadline constraint. The selective offloading technique will schedule the transfer of a certain amount of data for each time slot if executing the task remotely. It will schedule the CPU frequency

for each time slot if the task is executed locally.

Task Model

A model that completely depicts all the aspects of a mobile application is complex (Zhang *et al.*, 2013). Many details must be considered. However, such a high level of detail could render the problem mathematically intractable without offering meaningful insights for engineering practice. They proposed a classic model that captures the base of a typical mobile application. Specifically, a mobile application is abstracted into a profile with the following two parameters:

• Input data size S: the data length of the job.

• Application completion deadline t_D: the deadline constraint before which the application should be completed. Notice that the input data size S and the application completion deadline t_D impact mobile applications' energy consumption. Normally, with more data input and/or shorter completion deadlines, the energy consumption can be higher. Accordingly, we use these two parameters to capture the mobile applications' features and denote the application profile as $A(S, t_D)$.

Local Execution

The clock frequency of the chip, f, is approximately linear in proportion to the voltage supply of energy per operation and can be expressed as:

$$Ew(f) = kf^2 \tag{3.1}$$

where k is the effective switched capacitance depending on the chip architecture.

The task workload is calculated by the number of CPU cycles required by the application, denoted as W, determined by input data size and complexity of the algorithm in the application. Under optimal CPU frequency scheduling, the minimum energy consumption for the mobile execution is given by:

$$\mathcal{E}_m^* = \min_{\psi \in \Psi} \{\mathcal{E}_m(S, t_D, \psi)\} \tag{3.2}$$

Remote Execution

where ψ = f_1, f_2, ...f_W is any clock-frequency vector that meets the deadline constraint, ψ is the set of all feasible clock-frequency vectors, and $E_m(S, t_D, \psi)$ is the total energy consumed by the mobile device.

For a wireless fading channel with a gain of g, the energy consumed for transfer of S bits of data over the channel within a time slot can be represented as a convex monomial function:

$$\mathcal{E}_t(s, g, n) = \lambda \frac{s^n}{g} \tag{3.3}$$

where n denotes the monomial order and λ denotes the energy coefficient. This monomial function has been widely used. For the cloud execution, its total energy cost can be minimized by optimally adjusting the data rate (the number of transmitted bits in a given time slot) in response to a random channel.

With optimal transmission scheduling, the minimum cost of transmission energy for the remote execution is given by

$$\mathcal{E}_c^* = \min_{\phi \in \Phi} E\{\mathcal{E}_c(S, T, \phi)\} \tag{3.4}$$

Channel Model

They adopt the Gilbert-Elliott (GE) channel model as a stochastic model for practical circumstances in cloud execution. The GE channel model has two states — "good" and "bad" channel conditions — referred to as G and B. The two states represent two channel gain grades. If the measured channel gain is above some value, the channel is known as good; otherwise, the channel is known as bad. g_G and g_B denoted the channel gain for two statuses.

In the GE channel model, the state transition matrix is completely determined by the values of p_{GG} and p_{BB} .p_{XY} denoted the probability for the current state as X, and the next timeslot's channel state is Y. Accordingly, we have $p_{GB} = 1p_{GG}$ and $p_{BG} = 1p_{BB}$, where p_{GB} denotes the probability that the channel will transit from the good state to the bad state in the next time slot and p_{BG} denotes the probability that the channel will transit from the bad state to the good state in the next time slot. The state sojourn time is geometrically distributed. As such, the mean state sojourn time, measured in number of time slots in the good or bad state, is given by $T_G = 1/(1pGG)$ and $T_B = 1/(1p_{BB})$.

Problem Formulation

We must decide where to execute the application to minimize the total energy consumed on the mobile device(Zhang *et al.*, 2013). Specifically, the optimal policy is determined by the following decision rule:

$$\text{Mobile Execution} \ \text{ if } \mathcal{E}_m^* \leq \mathcal{E}_c^* \ \text{Cloud Execution} \quad \text{if } \mathcal{E}_m^* > \mathcal{E}_c^*. \tag{3.5}$$

$\mathcal{E}_m^*$ is proportional to and $\mathcal{E}_c^*$ is proportional to λ. The ratio between k and λ determines

the execution location.

The optional solution for the data transmission schedule via remote execution can be given by:

$$\min_{s_t}: \quad E\left[\sum_{t=1}^{T}\mathcal{E}_t(s_t,g_t)\right] \text{s.t.}: \quad \sum_{t=1}^{T}s_t = S, \quad s_t \geq 0, \forall t. \tag{3.6}$$

The minimum expected energy depends on the channel state at $t = T+1$($g_{T+1} = G$ or $g_{T+1} = B$). l_t is the number of bits not yet completed, and the optimal schedule is written by:

$$s_t^*(l_t,g_t;G) = l_t\left[\frac{(g_t)^{\frac{1}{n-1}}}{(g_t)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1G}})^{\frac{1}{n-1}}}\right], \ t \geq 2, l_1, \qquad t = 1, \tag{3.7}$$

If $g_{T+1} = G$, we have:

$$\zeta_{t;G} = p_{GG}\left[\left(\frac{1}{(g_G)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1;G}})^{\frac{1}{n-1}}}\right)^{n-1}\right]+p_{GB}\left[\left(\frac{1}{(g_B)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1;G}})^{\frac{1}{n-1}}}\right)^{n-1}\right], \ t \geq 2,$$

$$\text{p}_{GG}\left[\tfrac{1}{g_G}\right]+p_{GB}\left[\tfrac{1}{g_B}\right], \qquad t = 1 \tag{3.8}$$

Then:

$$s_t^*(l_t,g_t;B) = l_t\left(\frac{(g_t)^{\frac{1}{n-1}}}{(g_t)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1B}})^{\frac{1}{n-1}}}\right), \ t \geq 2, l_1, t = 1 \tag{3.9}$$

$$\zeta_{t;B} = p_{BB}\left[\left(\frac{1}{(g_B)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1;B}})^{\frac{1}{n-1}}}\right)^{n-1}\right]+p_{BG}\left[\left(\frac{1}{(g_G)^{\frac{1}{n-1}}+(\frac{1}{\zeta_{t-1;B}})^{\frac{1}{n-1}}}\right)^{n-1}\right], \ t \geq 2,$$

$$\text{p}_{BB}\left[\tfrac{1}{g_B}\right]+p_{BG}\left[\tfrac{1}{g_G}\right], \qquad t = 1 \tag{3.10}$$

If $g_{T+1} = B$, correspondingly, the minimum transmission energy is:

$$\mathcal{E}_c^*(L, T; G) = \lambda L^n \zeta_{T,G} \tag{3.11}$$

$$\mathcal{E}_c^*(L, T; B) = \lambda L^n \zeta_{T;B} \tag{3.12}$$

The minimized transmission energy is given by:

$$\mathcal{E}_c^*(S, T) = \frac{T_G}{T_G + T_B} \mathcal{E}_c^*(S, T; G) + \frac{T_B}{T_G + T_B} \mathcal{E}_c^*(S, T; B) \tag{3.13}$$

3.2.2 Collaborative Offloading

In this subsection, we will give some explanations for collaborative task execution both locally and remotely under the soft deadline constraint. In the collaborative offloading technique, the task will be split into multiple parts, where each part can be executed remotely or locally.

Task Model

They assume a mobile application is presented via a sequence of tasks with a linear topology figure 3.1. There are n tasks in the mobile application. Each task is sequentially executed, with output data as the input of the subsequent task (Zhang *et al.*, 2014). To describe the parametric context of each task, we define a tuple representation as ϕ_k = (w_k, α_k, β_k), where k = 1, 2, . . . , n. Specifically, ωk is the computing workload of the kth task. As such, the total computing workload of the application is $\sum_{k=1}^{n} w_k$. We denote the input and output data size of the kth task as k and β_k, respectively. Notice that the output of the $(k-1)$th task is the input of the kth task, i.e., $alpha_k$ = β_k1.

In this paper, we focus on the collaborative task execution between mobile devices and

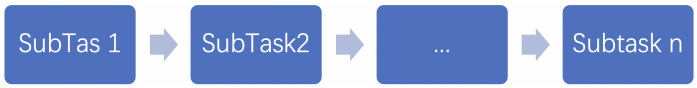

Figure 3.1: A Simple Workflow for the Multi-Part Offloading

cloud clones. Specifically, each task in the linear topology can be executed on the mobile device or offloaded to the cloud clone for execution based on the context of the task ϕ and the data rate R_t.

Local Execution

If the kth task is executed on the mobile device, the completion time is given by

$$d_m(k) = \omega_k f_m^{-1} \tag{3.14}$$

and the energy consumption of the mobile device is given by

$$e_m(k) = d_m(k) p_m \tag{3.15}$$

where f_m and p_m are the clock frequency and the computation power of the mobile device. We assume that p_m is fixed and does not change during the computation.

Remote Execution

If the kth task is executed on the cloud clone, the mobile device is idle and the wireless network interface card is turned off during the cloud execution. We denote $dc(k)$ as the completion time of the kth task executed on the cloud, given by

$$d_c(k) = \omega_k f_c^{-1} \tag{3.16}$$

where f_c is the clock frequency of the processing unit on the cloud clone, assumed to be faster than the CPU, the clock frequency of the mobile device (i.e., f_c ¿ f_m). In this case, we have $dc(k)$ ¡ $dm(k)$. Correspondingly, the energy consumption of the mobile device is given by

$$e_c(k) = d_c(k)p_i \tag{3.17}$$

where p_i is the power of the mobile device when idle. We assume that $p_i < p_m$ and hence $ec(k) < em(k)$.

• Sending Input Data (SID): If the kth task is offloaded to the cloud for execution, the input data α_k is sent to the cloud before execution. We denote ds(k) as the transmission time. Supposing the current time slot is i, we have

$$d_s(k) = \min\left\{ j : \sum_{t=i}^{i+j-1} R_t \geq \alpha_k \right\} \tag{3.18}$$

and the energy consumption of the mobile device is given by

$$e_s(k) = d_s(k)p_s \tag{3.19}$$

Channel Model

For the Markovian stochastic channel, they use the special case of the Markovian channel model as we mentioned before. They used the Gilbert–Elliott (GE) channel model, where there are two states: ‘good’ and ‘bad’ channel conditions (denoted as G and B respectively). The two states correspond to a two-level channel gain.

The transit matrix is given by:

$$P = \begin{bmatrix} pBB & pBG \\ pGB & pGG \end{bmatrix}. \tag{3.20}$$

Problem formulation

They formulate the optimal energy collaborative task execution as a constrained shortest path problem. The collaborative task execution between the mobile device and the cloud can be modeled by a directed acyclic graph (DAG) $G = (V, A)$, with the finite node-set, denoted as V, and arc set denoted as A. We denote the number of nodes and arcs as $|V|$ and $|A|$ respectively. At the start, they introduce two dummy nodes (i.e., node S) as the source node for application initiation and node D as the destination node for application termination respectively. Node k represents that the kth task has been completed on the mobile device, while node k represents that the kth task has been completed by the cloud clone, where k = 1, 2, . . . , n. We can find that $|V| = 2n + 2$ and $|A| = 4n$ (Zhang *et al.*, 2014).

3.3 Mobile Computation Offloading Under A Hard Deadline Constraint

In this section, we will give details about three offloading techniques under the hard deadline constraint. The hard deadline constraint must be satisfied to be more strict. We will give details including the task model; the energy and time model for the local and remote execution; the channel model; and the problem formulation for continuous offloading, multi-part offloading, and preemptive offloading.

3.3.1 Continuous Offloading

In this subsection, we will explain the task execution locally and remotely under the hard deadline constraint with the continuous offloading methods.

Task Model

We consider the execution of computational tasks (jobs) generated by a mobile device, either locally (by the device itself) or remotely (by offloading them on a remote cloud server through a wireless transmission channel) (Hekmati *et al.*, 2019a). We focus on a single task with characteristics known at its release time. Time values are thus referred to by their time slot indices.

t_r: Release time of the job, i.e., the time when the job is ready to start execution, either locally or via offloading. For convenience, we assume t_r = 1.

t_D: Hard deadline of the job, i.e., the job execution results must be available on the mobile device by time t_D, where T_D = t_D - t_r - 1 is the maximum number of time slots available for completing the job.

S_{up}: Number of bits transmitted through the uplink channel when uploading the job to the cloud.

S_{down}: Number of bits transmitted through the downlink channel when downloading job results from the cloud.

Local Execution

The energy cost and time needed to execute a job locally are assumed to be known at the job release time, t_r, and are defined by E_L and T_L, respectively. If the computation offloading algorithm elects to execute the job locally without remote offloading, we must ensure the

job deadline is always satisfied. Therefore, local execution must start no later than:

$$t_L = t_D - T_L + 1 \tag{3.21}$$

Remote Execution

We will assume that, upon its release, the job is assigned an execution time T_{exec} by the cloud server, which is communicated to the mobile device. If T_{up} and T_{down} are the time periods needed to upload the job to the cloud server and download its results to the device, respectively, the total offloading time T_{off} is given by:

$$T_{off} = T_{up} + T_{exec} + T_{down} \tag{3.22}$$

The channel is assumed to use bit rate adaptation to accommodate random variations in channel conditions. Thus, T_{up} is a random variable dependent on uplink channel state evolution as a given upload occurs. In what follows, it is assumed that the channel state can be modeled as a homogeneous discrete-time Markov process; the same holds for T_{down}.

Channel Model And Problem Formulation

CSMC is memoryless as far as the state of offloading and channel conditions are concerned; to incorporate them into our model, we form a new Markov chain, referred to as a time-dilated absorbing Markov chain (TDAMC). In the evolution of the system starting at the current time slot t_s and running until the computation has been completed, either locally or remotely, the state of the channel in each TDAMC state at time $t >= t_s$ is represented by X_t, where X_t is a member of M. However, unlike the CSMC, the TDAMC incorporates t and other information into its structure, which can reflect the channel status with the time

series.

The TDAMC models the job offloading progress if the latter is initiated at the current time slot t_s. It is a rooted tree, constructed as follows: The root state is the channel state X_{ts} at current time slot t_s; since this is the current time slot, X_{ts} is known (Figure 3.2). At each subsequent time slot, the Markov chain tree branches forward according to the transitions possible from the current state (X_{ts}, initially) to other CSMC states. At each step along a given tree branch, the number of job bits transmitted is determined by the bit rate associated with the channel state in question.

This construction continues along each branching tree path until the number of bits offloaded reaches the job upload size, S = S_{up}. At that point, the state reached in the TDAMC is defined as a Markov chain absorbing state, i.e., it has a self-transition with a probability of 1. From this construction, it can be seen that the TDAMC includes all possible paths that lead to a successful job offload and that all of the states are either transient or absorbing. Eventually, all paths terminate in an absorbing state, and the path's energy cost is proportional to its length.

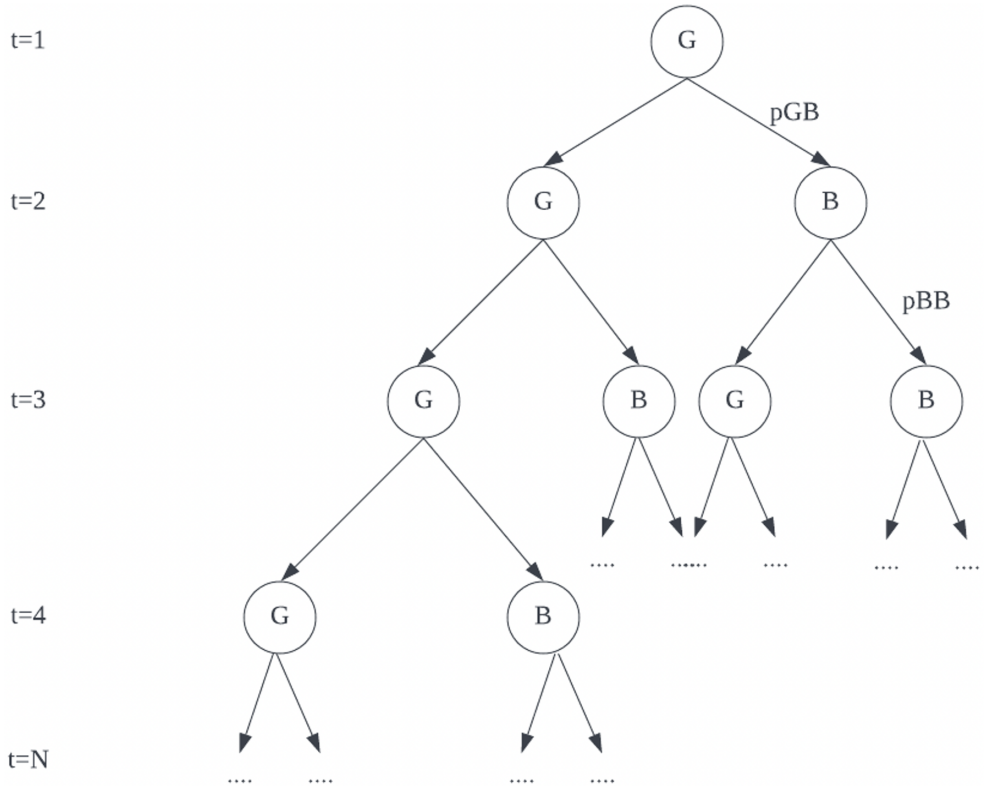

Figure 3.2: A example TDAMC that builds from the GE model (Hekmati *et al.*, 2019a)

3.3.2 Multi-Part Offloading

This subsection will explain task execution locally and remotely under the hard deadline constraint with the multi-part offloading technique. This technique will split the task into a specific number of parts, upload it to the remote server during offloading, and execute the entire task after completing the upload progress.

Task Model

Multi-part offloading is similar to continuous offloading on the offload decision, but it sends the task to the remote server in multiple parts. The number is determined by the parameter

K (Hekmati *et al.*, 2019b).

Local Execution

A job's local execution energy consumption determines its CPU workload. We assume that the local execution energy consumption is E_L, and the local execution time is T_L.

We must ensure that the task deadline constraint is always satisfied. Therefore, local execution must start T_L time slots before the job deadline. If remote execution results have not arrived at t_L:

$$t_L = t_D - T_L + 1 \tag{3.23}$$

Remote Execution

We assume that the data are split into two parts transmitted to the server individually. Uploading the first and second part takes T_{up1} and T_{up2} time slots, respectively; T_W is the waiting time between the two uploads. When it's released, the remote server assigns the job a server execution time T_{exec} and a resulting downloading time T_{down} — which are communicated to the mobile device — and their total time is:

$$T_{rest} = T_{exec} + T_{down} \tag{3.24}$$

So, the total offloading time is:

$$T_{off} = T_{up1} + T_W + T_{up2} + T_{exec} + T_{down} \tag{3.25}$$

It is assumed that the mobile device transmits a fixed power and uses bit-rate adaptation

to accommodate random variations in the uplink channel conditions. As a result, T_{up1} and T_{up2} are random variables dependent on the evolution of the uplink channel state as a given upload occurs. In what follows, it is assumed that the channel state can be modelled as a homogeneous discrete-time Markov process.

Channel Model And Problem Formulation

Like the channel model in the continued offloading, they also use the TDAMC as the channel model in this paper. Assuming that there is a known channel - state Markov chain (CSMC), each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded in that state (Hekmati *et al.*, 2019b). The CSMC transition matrix is defined as P = P[i,j], where P_{ij} is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. As defined, the CSMC is memoryless, but we will need to incorporate time into it in what follows. Therefore, we will consider the tree-like Markov chain given by following the evolution of the channel, starting from an initial state at time $t = 1$ and branching out from each state according to the transition probabilities of the CSMC.

3.3.3 Preemptive Offloading

In this subsection, we will explain the task execution locally and remotely under the hard deadline constraint with the preemptive offloading technique. This technique will split the task into multiple parts according to the wireless channel environment, upload it to the remote server during the offloading, and then execute it at one time after completing the upload progress.

Task Model

Each job to be executed is characterized by the following(Teymoori *et al.*, 2021): t_R: The job's release time, i.e., when the job is ready to start execution, either locally or via offloading. For convenience, we will assume that t_R = 1. t_D: The job's hard deadline, i.e., the job execution results must be available on the mobile device by time t_D. $T_D = t_D - t_R + 1$ is the maximum number of time slots available for completing the job.

Local Execution

The preemptive offloading decides according to different parameters(Teymoori *et al.*, 2021). It also will split the task into multiple subtasks, but the number and size of the parts are not given; it will adjust according to the wireless channel environment. If the computation offloading algorithm elects to execute the job locally, we must ensure that the job deadline is always satisfied. Therefore, local execution must start by:

$$t_L = t_D - T_L + 1 \tag{3.26}$$

T_L can be calculated when the task is released. Hence, according to the CLE model we are proposing, an overlapping of the local and remote executions of the task may occur, but the hard job deadline is always respected, even if channel contentions or extended channel outages occur. We assume that a fraction of the local CPU is dedicated to processing the task, such that the local processing speed is fixed and known, and T_L can be calculated when the task is released. In the mobile energy consumption formulations described above and in the next section, we have excluded the local energy requirements needed to execute the online scheduling algorithms.

Remote Execution

Regarding job offloading, we will assume that upon a job's release, the cloud server assigns it an execution time T_{exec}, which is communicated to the mobile device (or prescribed by the contractual agreement between the device's user and the cloud server operator). Additionally, we assume that the user has been allocated capacity (like recurring time slots) until the offload has been completed. The total offloading time T_{off} is given by:

$$T_{off} = t_{o_n} t_{o1} + 1 + T_{exec} + T_{down} \tag{3.27}$$

Then the T_{rest} are given by:

$$T_{rest} = T_{exec} + T_{down} \tag{3.28}$$

It is assumed that the channel uses bit rate adaptation to accommodate random variations in channel conditions. Consequently, T_{off} is a random variable, dependent on the uplink channel state's evolution during a given upload. It is assumed that the channel state can be modeled as a homogeneous discrete-time Markov process.

Channel Model And Problem Formulation

All offloading deadlines, job sizes (in bits), and energy costs are supposedly related only to job uploading; we assume that T_{exec} and T_{down} are known and already incorporated into the energy costs and the job deadline. Given the ensuing results, adding the effects of T_{exec} and T_{down} is straightforward. As mentioned above, we assume that CSMC is known (i.e., the channel conditions evolve from a one-time slot to the next according to a homogeneous finite state Markov chain). Like the previous offloading mechanism, each

state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded in that state. The CSMC transition matrix is defined as P = P[i,j], where P[i,j] is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. As defined, CSMC is memoryless, but in what follows, we will need to incorporate time into it, the already offloaded bits, and the decision of uploading or not at every time slot. Therefore, we construct a new Markov decision process, the time-dilated Markov process (TDMP), as follows: For every time $1tt_D + 1$, we define a set of states (X_t, S_t), where X_t is a channel state and $0S_tS_{up}$. We can consider the states arranged in layers for $t = 1, 2, ..., t_D + 1$. The set of actions contains two actions, a_0 and a_1, corresponding to not offloading and offloading, respectively.

Chapter 4

Optimal Solutions

4.1 Introduction

To conduct our tests and compare them more objectively, we will outline various modifications and approaches to each method in this part and some initial descriptions of the algorithms.

4.2 Mobile Computation Offloading Under A Soft-Deadline Constraint

In this section, we will give the details about how the offloading decision algorithm works in the two soft deadlines offloading techniques - selective offloading as well as collaborative offloading. We will also present the pseudo-code for the algorithms we will use in the experiments.

4.2.1 Selective offloading

While offloading, this task considers the CPU's rapid changes and wireless power fluctuations, frequently modifying the CPU per the processes' demands using the DVFS (Zhang *et al.*, 2013). It works well with most portable electronics. This technique aims to determine the best offloading choice in single-user scenarios with a soft deadline restriction. These techniques, however, are unrestricted to the CPU power and transmission power during the offload. When compared to other algorithms, unrestricted CPU power and transmission power significantly differs. Other methods typically set a predetermined limit on the mobile device's power. To assess how well the decision algorithms perform compared to other algorithms, we frequently offer them the CPU and a limited amount of power for wireless transfer.(Zhang *et al.*, 2013)

4.2.2 Collaborative Offloading

Yonggang Wen and Weiwen Zhang (Zhang *et al.*, 2014) proposed the algorithm, which considers the single-user and soft-deadline constraints like the previous methods. It is a multi-part offloading algorithm for the soft-deadline restriction mentioned in Chapter 3. Compared to the initial algorithms in Chapter 4, it is more sophisticated. The entire task was divided into several linear subtasks. A linear topology is used to model the subtask:

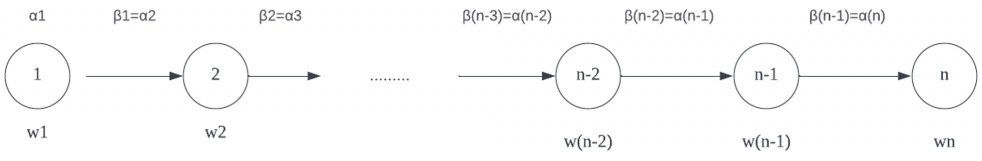

Figure 4.1: A topology for the multipart offloading (Zhang *et al.*, 2014)

The α_i represents the S_{ini} , and the β_i represents the S_{out} aforementioned. Each task

also has its own workload W_i, measured in the number of CPU cycles. The whole offloading progress can be represented as a DAG graph like below Figure 4.2.

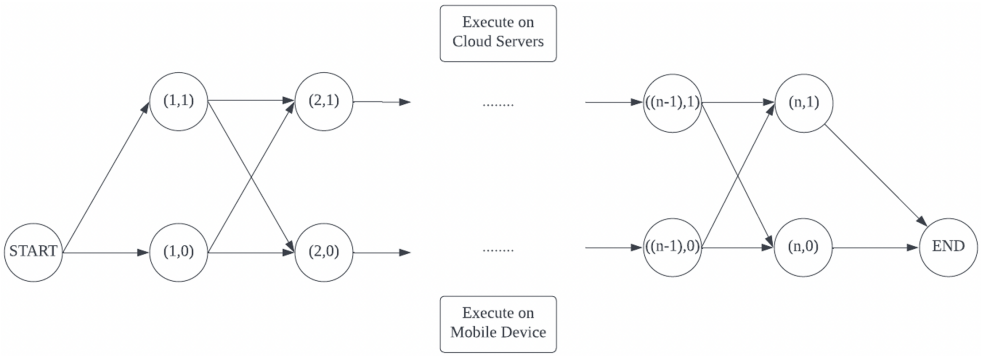

Figure 4.2: A DAG graph for the Multipart offload (Zhang *et al.*, 2014)

And the following are the LARAC algorithms for executing collaborative tasks with low-energy-consumption algorithms 1.

Algorithm 1 The adapted LARAC algorithm for the energy-efficient collaborative task execution

Input: $G(V, A)$

Output: p_λ^*

1: //find the shortest path in the graph of energy consumption; we use the Dijkstra in this thesis

2: $p_e = \mathrm{FindShortestPath}(S, D, e)$

3: //if p_e can meet the time delay, then p_e is optimal, return directly

4: **if** $d(p_e) \leq T_d$ **then**

5: return p_e

6: **end if**

7: //find the shortest path regarding time delay

8: $p_d = \mathrm{FindShortestPath}(S, D, d)$

9: //if p_d cannot meet the time delay, then there is no solution
10: **if** $d(p_d) > T_d$ **then**
11: returns null, exit directly
12: **end if**
13: **while** true **do**
14: //obtain λ^* by updating p_e and p_d
15: $\lambda = (e(p_e) - e(p_d)) / (d(p_d) - d(p_e))$
16: //find the shortest path in the graph of the aggregated cost
17: $p_\lambda = \mathrm{FindShortestPath}(S, D, e_\lambda)$
18: **if** $e_\lambda(p_\lambda) = e_\lambda(p_e)$ **then**
19: return p_d
20: **else**
21: **if** $d(p_\lambda) \leq T_d$ **then**
22: $p_d = p_\lambda$
23: **else**
24: $p_e = p_\lambda$
25: **end if**
26: **end if**
27: **end while**

The study transforms the offloading choice problem into a more typical computer science problem: locating the shortest path in the graph, where the cost on the given graph equals the path's entire edge weight. Two DAG-like graphs, similar to those in Figure 4.2, serve as the algorithms' input.

The edges are referred to as the cost for the status transfer; the node is referred to as the status. A single graph represents the time or delay cost, and the edge's weight reflects

the expense of changing from one status to another. Another graph depicts the energy cost. The edge's weight in the graph reflects energy costs for moving from one status to another. The methods start by finding the shortest path on the p_e energy graph, which stands for the solution requiring the least amount of energy without considering the delay restriction and then checking if the solution could meet the delay constraint. The methods return the answer immediately if the delay constraint can be met . Then, by locating the shortest path on the delay graph, it will identify the solution requiring the least amount of time to complete the entire work. If the minimized delay cannot satisfy the delay constraint, it will immediately return an error and terminate the procedures. After that, a while loop will begin. Every cycle of the loop will compute λ by the given equation:

$$\lambda = \frac{e\left(p_e\right) - e\left(p_d\right)}{d\left(p_e\right) - d\left(p_d\right)} \tag{4.1}$$

And generate a new graph by aggregating two graphs with the parameter λ. The cost for p_e and p_d on two graphs determines the λ. Furthermore, determine the minimal aggregated by determining the shortest path on the combined graph utilizing the shortest path techniques on the new graph. If the aggregated minimum cost equals the cost of path p_e on the aggregated graph, stop the algorithms. If not, modify p_e and p_d before beginning the subsequent cycles based on whether the delay limitation satisfies the delay cost on p_e. The Dijkstra algorithms serve as the shortest path algorithms in our experiments.

4.3 Mobile Computation Offloading Under A Hard Deadline Constraint

In this section, we will give the details about how the offloading decision algorithm works in the three hard deadlines offloading techniques - continuous offloading, multi-part offloading as well as collaborative offloading. We also gave the pseudo-code for the algorithms we will use in the experiments.

4.3.1 Continuous Offloading

According to the hard deadline restriction by Arvin Hekmati, Peyvand Teymoori, and Terence D. Todd, a continuing offloading decision method is used (Hekmati *et al.*, 2019a) because this algorithm's deadline restriction is a hard deadline, implying the time restrictions will always be met. The graph that follows indicates the task computation time:

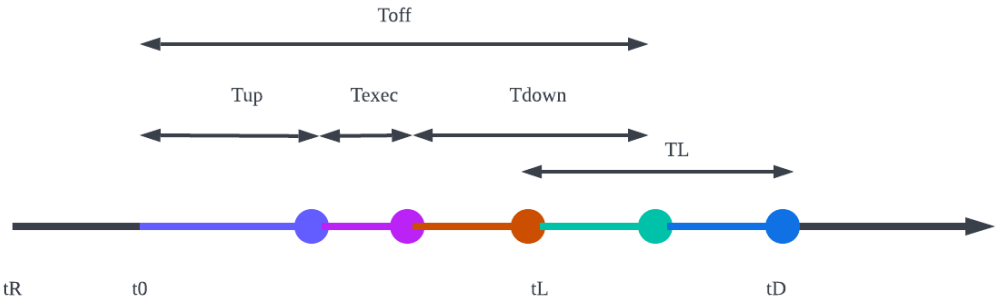

Figure 4.3: The timeline for the OnOpt(Hekmati *et al.*, 2019a)

As mentioned, the local execution must start no later than:

$$t_L = t_D - T_L + 1 \tag{4.2}$$

The pseudo-code for the OnOpt is given by:

Algorithm 2 OnOpt (Online Optimal) Algorithm

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job size S.

1: $t_o := \infty$ ▷

Offloading initially disabled (t_o is offloaded start time)

2: **for all** $t_s \in \{1, \ldots, t_D\}$ **do**

3: **if** $t_s < t_o$ **then**

4: $\overline{c_{t_s}} := g_{t_s}(x)$ ▷

calculate the expected energy cost of offloading at t_s.

5: $\overline{c_{t_s+1}} := E\left[V_{t_s+1} \mid X_{t_s} = x\right]$ ▷

calculate expected cost of waiting until $t_s + 1$.

6: **if** $\overline{C_{t_s}} \leq \overline{C_{t_s+1}}$ **then**

7: $t_o := t_s$ ▷

Start offloading.

8: **end if**

9: **else**

10: **if** offloading end at t_s **then**

11: if the local execution is running; abort it. ▷

The response from the remote server has been received, and the progress ends.

12: **return**

13: **end if**

14: **if** $t_s = t_D - T_L + 1$ **then**

15: Start local execution. ▷

The local execution will execute with the offload concurrent to ensure the deadline is satisfied.

end if

if $t_s = t_D$ **then**

if the remote execution is running; abort it ▷

Local execution has completed.

return

end if

end if

end for

In the algorithms, if the offloading has never been started before for each time slot, it will determine whether the current time slot's projected total energy consumption is less than the following time slot's expected total energy consumption before beginning the offloading. If offloading has already begun, determine whether it has been completed; if so, stop local execution if the current time is later than t_L. When the task is finished, the algorithms will then terminate. Start the local execution if the time is at t_L and the task hasn't been finished. If the t_D is reached, remote offloading will stop. The task was also finished by the local execution simultaneously. In other words, if the unload response is not received, the local execution will begin at t_L, even though the remote execution has already begun. To ensure the strict timeline limitation is met, the local and remote execution will occur simultaneously.

In the GE model, b is the number of Bad state time slots that occur during offloading, and T_{off} is the amount of time needed to finish the offloading process. The bit rates for Good status and Bad status are r_G and r_B, respectively. The bit rate for two statuses determines the 'b' limitation, the size S:

$$\frac{(T_{off}-1)\,r_G - S}{r_G - r_B} < b \leq \frac{T_{off} r_G - S}{r_G - r_B} \tag{4.3}$$

Of which b is just an integer.

The relationship between the number of the Bad status during the offloaded multiplied by the rate of Bad status and the number of Good status during the offloaded multiplied with the rate of Good status can be determined for every b satisfying the preceding constraint:

$$\hat{S} = b * r_B + (T_{off} - b) * r_G \tag{4.4}$$

And then, assume that X_t is the channel status for the current time slot t with the probability of the status transition. The correspond $\widehat{P}_t\left(T_{off}, b, X_t\right)$ for the b can be calculated by:

When the $\hat{S} - S \geq r_B$ and the channel status X_t is Good:

$$\hat{P}_t(T_{off}, b, X_t) = \begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{4.5}$$

When the $\hat{S} - S < r_B$ and the channel status X_t is Good:

$$\hat{P}_t(T_{off}, b, X_t) = \begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} + \\ + \sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k+1} P_{BG}^{k} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{4.6}$$

When the $\hat{S} - S \geq r_B$ and the channel status X_t is Bad:

$$\hat{P}_t(T_{off}, b, X_t) = \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k} \binom{T_{off}-b-1}{k} P_{GB}^k P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} \tag{4.7}$$

When the $\hat{S} - S < r_B$ and the channel status X_t is Bad:

$$\begin{aligned}\hat{P}_t(T_{off}, b, X_t) = &\sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k} \binom{T_{off}-b-1}{k} P_{GB}^k P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} \\ &+ \sum_{k=1}^{\min(b-1, T_{off}-b)} \binom{b-1}{k} \binom{T_{off}-b-1}{k-1} P_{GB}^k P_{BG}^k P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k}\end{aligned} \tag{4.8}$$

Then, regarding the current channel status given X_t and the T_{off}, we can calculate the $P_t\left(T_{off}, X_t\right)$ by sum all $\widehat{P}_t\left(T_{off}, b, X_t\right)$ for the b, satisfying the constraint. Refer the B_min is r_B and the $B)max$ is r_G, the offloading starts at t and the channel status at x. The expected energy consumption for offloading $E_{off}\left(t, x\right)$ and the expect energy consumption $E_L\left(t, x\right)$ by:

$$E_{off}(t,x) = \begin{cases} E_{tr} \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off}, x) T_{off}, & 1 \leq t < t_D - \frac{S}{B_{min}} + 1 \\ E_{tr} \left(\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{t_D - t} P_t(T_{off}, x) T_{off} + \sum\limits_{T_{off}=t_D-t+1}^{\frac{S}{B_{min}}} P_t(T_{off}, x)(t_D - t + 1) \right), & t_D - \frac{S}{B_{min}} + 1 \leq t \leq t_D \\ 0, & t > t_D \end{cases} \tag{4.9}$$

$$E_L(t,x) = \begin{cases} \sum\limits_{T_{off}=t_L-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & 1 \le t < t_L \\ \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & t_L \le t \le t_D \\ E_L & t > t_D \end{cases} \tag{4.10}$$

And the total expected energy consumption for the given time and the $\overline{C_{ts}}$ in the OnOpt algorithms can be given by:

$g_t(x) = E_{off}(t,x) + E_L(t,x)$

Then, to calculate the expected energy consumption for waiting until the next time slot $\overline{C_{ts+1}}$ in the algorithm, we can calculate it recursively step-by-step according to the given equation:

$$V_t(x) = \begin{cases} E_L, & t \ge t_D \\ \min\{g_t(x), E[V_{t+1}|X_t = x]\}, & t = t_s, \cdots, t_D - 1 \end{cases} \tag{4.11}$$

$$E[V_{t+1}|X_t = x] = \sum_{y \in M} Pr[X_{t+1} = y|X_t = x]V_{t+1}(y) \tag{4.12}$$

Although we can obtain the solution utilizing recursive methods directly, the time complexity is too high. Furthermore, this recursive solution has a plethora of identical sub-problems. Therefore, we can lessen the complexity of time by employing the DP (Dynamic Programming). By adding an array Cache[x][t] to store the solutions to each sub-problem while the process runs, we can 'update' the recursive algorithms the DP algorithms use. The t is the index of the time slots, and the x represents the status, which can lower the time

complexity to roughly O(n^2) by optimizing the cache O(n^n) — far less than the recursive origin's time complexity O(n^n)) .

4.3.2 Multi-Part Offloading

Arvin Hekmati and Peyvand Teymoori present a multi-part unloading (Hekmati *et al.*, 2019b) — an enhanced version of 'OnOpt' dividing a single task into numerous parts, just like collaborative task execution. However, unlike collaborative task execution, each portion of the task can be agilely chosen and carried out on a local device or a remote server. The entire task must be completed in one location. Only the task itself can be divided into various parts for upload.

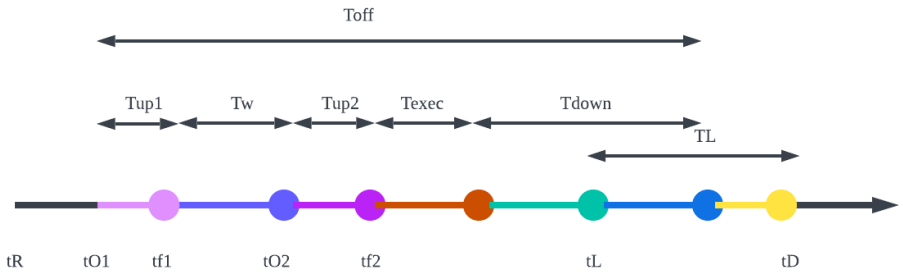

Figure 4.4: The time line of MultiOpt(Hekmati *et al.*, 2019b)

Figure 4.4 depicts the time graph for the MultiOpt. Assumably, the assignment has been divided into two pieces for upload. The two components are S_{up1} and S_{up2} in size. The time spent on the two segments is T_{up1} and T_{up2}. It will have time to wait between the two upload progresses to locate a better opportunity to upload the following parts after the previous parts have finished uploading T_w. After all task components have been properly uploaded, the execution will begin on the remote server. Offloading the task to the remote server takes time as follow:

$$T_{off} = T_{up1} + T_w + T_{up2} + T_{exec} + T_{down} \tag{4.13}$$

The T_{rest} is the sum of the T_{exec} and T_{down}:

$$T_{rest} = T_{exec} + T_{down} \tag{4.14}$$

We must determine the absorbing probability matrix for each part of the MultiOpt algorithms per the provided TDAMC. We only consider breaking the task into two halves during the upload process in our studies. The matrix of absorption probabilities $W1$ and $W2$ represents the absorption probability for two tasks. Whenever a portion reaches the absorbing status, it has already been posted. You may compute the absorption probabilities matrix as follows:

$$W = NR \tag{4.15}$$

$$N = (I - Q)^{-} 1 \tag{4.16}$$

R is the matrix for the probability from the transient status to the absorbing status. Q is the matrix containing the probability of the transient status. With $W1$ and $W2$, we can calculate the probability of P_{t01} and P_{t02} according to the given size S_{up1} and S_{up2}. Assume we start the offload at X_{to1} and X_{to2} at T_{o1} and T_{o2}.

$$P_{to1}\left(S_{up1}, T_{up1}, X_{to1}\right) = \sum_{j \in S} W_1\left[X_{to1}, j\right] \tag{4.17}$$

$$P_{to2}\left(S_{up2}, T_{up2}, X_{to2}\right) = \sum_{j \in S} W_2\left[X_{to2}, j\right] \tag{4.18}$$

This means the probability of completing the uploading of the parts with size S starts

from T_o exact consume T_{up} time slots.

The GE model, a TDAMC with only two status, is the channel model we utilize in our tests. They anticipate that the energy required to upload the two components of the task can be computed as follows:

$$E_{off1}\left(S_{up1}, X_{to1}\right) = \begin{cases} E_{tr}\sum_{T_{up1}=S_{up1}/r_G}^{S_{up1}/r_B} P_{to1}\left(S_{up1}, T_{up1}, X_{to1}\right) * T_{up1}, & \text{if } 1 \le t_{o1} < t_D - \dfrac{S_{up1}}{r_B} + 1 \\ E_{tr}\left(\begin{array}{c} \sum_{T_{up1}=\frac{S_{up1}}{r_G}}^{t_D - t_{o1}} P_{to1}\left(S_{up1}, T_{up1}, X_{to1}\right) * T_{up1} +, \\ \sum_{T_{up1}=t_D-t_{o1}+1}^{r_B} P_{to1}\left(S_{up1}, T_{up1}, X_{to1}\right) * \left(t_D - t_{o1} + 1\right) \end{array}\right), & \\ & \text{if } t_D - t_{o1} + 1 \le t_{o1} \le t_D \end{cases} \tag{4.19}$$

$$E_{L1}\left(S_{up1}, X_{to1}\right) = \begin{cases} \sum_{T_{up1}=t_L-t_{o1}+1}^{\frac{S_{up1}}{r_B}} P_{to1}\left(S_{up1}, T_{up1}, X_{to1}\right)\left(\dfrac{\min\left\{t_D+1, t_{o1}+T_{up1}\right\} - t_L}{T_L} * E_L\right), & \\ & \text{if } 1 \le t_{o1} < t_L \\ \sum_{T_{up1}=\frac{S_{up1}}{r_G}}^{r_{to1}}\left(S_{up1}, T_{up1}, X_{to1}\right)\left(\dfrac{\min\left\{t_D+1, t_{o1}+T_{up11}\right\} - t_L}{T_L} * E_L\right), & \\ & \text{if } t_L \le t_{o1} < t_D \end{cases} \tag{4.20}$$

$$\hat{E}_{off_2}\left(S_{up_2}, T_{up_2}, t_{o_2}\right) = \begin{cases} E_{tr}T_{up_2} + E_{rc}T_{down}, & t_{f_1} < t_{o_2} \le t_D - T_{up_2} - T_{rest} \\ E_{tr}T_{up_2} + E_{rc}\left\{t_D - \left(t_{o_2} + T_{up_2} + T_{exec}\right) + 1\right\}, & t_D - T_{rest} < t_{o_2} + T_{up_2} \le t_D - T_{exec} \\ E_{tr}T_{up_2}, & t_D - T_{exec} < t_{o_2} + T_{up_2} \le t_D \\ E_{tr}\left(t_D - t_{o_2} + 1\right), & t_D < t_{o2} + T_{up_2} \le t_D + T_{up_2} \end{cases} \tag{4.21}$$

$$\hat{E}_{L_2}\left(T_{up_2}, t_{f_1}, t_{o_2}\right)$$

$$= \begin{cases} 0, & t_{f_1} < t_{o_2} < t_L - T_{up_2} - T_{rest} \\ \dfrac{t_{o_2} + T_{up_2} + T_{rest} - t_L}{T_L} E_L, & t_{f_1} < t_L \wedge t_L - T_{up_2} - T_{rest} \leq t_{o_2} \leq t_D - T_{up_2} - T_{rest} \\ E_L, & t_{f_1} < t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \leq t_D \\ \dfrac{t_{o_2} + T_{up_2} + T_{rest} - t_{f_1}}{T_L} E_L, & t_{f_1} \geq t_L \wedge t_{f_1} < t_{o_2} \leq t_D - T_{up_2} - T_{rest} \\ \dfrac{t_D - t_{f_1}}{T_L} E_L, & t_{f_1} \geq t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \leq t_D \end{cases} \tag{4.22}$$

The pseudo-code for the MultiOpt if we set the K to 2 is:

Algorithm 3 MultiOpt (Multi-decision online Optimal) (Hekmati *et al.*, 2019b)

Input: Local execution starting time t_L, local execution energy E_L, j_{ob} deadline t_D, and j_{ob} sizes S_{up_1}, S_{up_2}.

1: **for all** $t = 1, \ldots, t_D$ **do**

2: Case 1: Checked if both parts are finished; if yes, exit directly

3: Case 2: If the uploading progress is running at t, continue.

4: Case 3: If uploading part 1 has not started, check according to the equation:

5: **if** min is g_1 then start uploading part 1. **then**

6:

$$V_1\left(X_{t_{o_1}}\right) = \begin{cases} E_L, & t_{o_1} \geq t_D \\ \min\left\{g_1\left(S_{up_1}, X_{t_{o_1}}\right), & t_{o_1} = 1, \ldots, t_D - 1 \\ E\left[V_1\left(X_{t_{o_1}+1}\right) \mid X_{t_{o_1}}\right]\right\}, & \end{cases}$$

7: **else**

8:

$$g_1\left(S_{up_1}, X_{t_{o_1}}\right) = E_{off_1}\left(S_{up_1}, X_{t_{o_1}}\right) + E_{L_1}\left(S_{up_1}, X_{t_{o_1}}\right) + \sum_{t_{f_1+1}\in\mathcal{S}_1} W_1\left[X_{t_{o_1}}, X_{t_{f_1}+1}\right] V_2\left(t_{f_1}, X_{t_{f_1}+1}\right).$$

9: **end if**

10: If part 1 has been uploaded but part 2 has not started uploading, then perform the check accordingly:

$$V_2\left(t_{f_1}, X_{t_{o_2}}\right) = \begin{cases} 0, \text{ if } t_{o_2} > t_{f_1} \geq t_D \\ E_L - \frac{\max\left\{t_{f_1}, t_L\right\} - t_L}{T_L} E_L, \text{ if } t_{o_2} \geq t_D > t_{f_1} \\ \min\left\{g_2\left(S_{up_2}, t_{f_1}, X_{t_{o_2}}\right), E\left[V_2\left(t_{f_1}, X_{t_{o_2}+1}\right) \mid X_{t_{o_2}}\right]\right\} \\ \quad \text{if } t_D > t_{o_2} \end{cases}$$

11: When min is g_2, then start uploading part 2. The g_2 can be given by:

$$g_2\left(S_{up_2}, t_{f_1}, X_{t_{o_2}}\right) = E_{off_2}\left(S_{up_2}, X_{t_{o_2}}\right) + E_{L_2}\left(S_{up_2}, t_{f_1}, X_{t_{o_2}}\right)$$

12: **end for**

Check the status of each part's upload for the algorithms in each time slot; if both parts have been uploaded, stop the algorithms. The cycle continues if certain portions are still being uploaded. If no parts are uploading, and there is at least one part that has not yet been posted, since part 1 must be uploaded before part 2 and if part 1 has not yet been done, start it now by determining whether the expected energy cost of offloading at the current time with the current channel status is minimal. If part 1 has already been uploaded and the current is inactive, determine whether the expected energy cost of offloading the second part at this time is minimal. If so, begin uploading the $S2$.

4.3.3 Preemptive Offloading

Water-Filling with Equilibrium (WF-Equ)

Peyvand Teymoori, Arvin Hekmati, and Terence D. Todd have proposed the Optimal Preemptive Offloading (OPO) (Teymoori *et al.*, 2021). Similarly, outsourcing decision-making algorithms like OnOpt must deal with hard-deadline constraints. However, to determine each time slot's expected energy consumption, the OPO algorithms must use dynamic programming (DP), which uses too many resources for calculations. Consequently, the WF-Equ has been proposed (Teymoori *et al.*, 2021).

Algorithm 4 Local execution starting time t_L, local execution energy E_L, job deadline t_D, equilibrium probabilities π, state space of the Markov chain $\mathcal{M} = \{1, 2, \ldots, m\}$, current state X_t, remaining size $S_r = S$.

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job size S.

for all $t = 1, \ldots, t_D$ **do**
  if $S_r = 0$ **then**
    break
  end if
  $\mathcal{F} = \{m\}$ $\triangleright$ m is the state with the highest bit rate; at the start, it only has one item
  $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$
  if $X_t \notin \mathcal{F}$ **then**
    $t_f = t + \dfrac{S_r}{B_{avg}} + T_{rest}$
  else
    $t_f = t + \dfrac{S_r - Br\left(X_t\right)}{B_{avg}} + T_{rest}$
  end if

while $(t_f > t_D)$ and $(\mathcal{M} \neq \mathcal{F})$ **do**

  choose the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ and add to $\mathcal{F}$

  $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$

  if $X_t \notin \mathcal{F}$ **then**

    $t_f = t + \dfrac{S_r}{B_{avg}} + T_{rest}$

  else

    $t_f = t + \dfrac{S_r - Br\left(X_t\right)}{B_{avg}} + T_{rest}$

  end if

end while

if $t_f > t_D$ **then**

  break

end if

if $X_t \notin \mathcal{F}$ **then**

  continue ▷ Continue with the next time slot $t + 1$

end if

$B_F = \dfrac{\sum_{i \in \mathcal{F}} \pi[i] Br[i]}{\sum_{i \in \mathcal{F}} \pi[i]}$

$E_{up} = \left(1 + \dfrac{\max\left\{0, S_r - Br\left(X_t\right)\right\}}{B_F}\right) E_{tr}$

$E_l = \left(\max\left(t_f + 1, t_L\right) - \max\left(t, t_L\right)\right) E_L / T_L$

if $t_f \leq t_D$ **then**

  $E_{off} = E_{up} + E_l + T_{down} E_{rc}$

else

  $E_{off} = \dfrac{t_D - \max\left\{t, t_L\right\} + 1}{T_L} E_L$

end if

if $E_{off} < \dfrac{t_D - \max\left\{t, t_L\right\} + 1}{T_L} E_L$ **then**

36: Perform offload at t

37: $S_r = \max\{0, S_r - Br(X_t)\}$

38: **end if**

39: **end for**

For a given transition probability matrix of the channel status in the GE model, the equilibrium probability can be calculated by solving the following equation:

$$(\pi_1, \pi_2) = (\pi_1, \pi_2) \begin{bmatrix} pBB & pBG \\ pGB & pGG \end{bmatrix} \tag{4.23}$$

The equilibrium probability of the relevant channel status transition probability matrix is the answer to the equation [π_1,π_2]. The water filling's progress is one of this algorithm's most crucial components. This algorithm is somewhat greedy. Each time slot is described regarding a set F—used to collect the status that has been added during the current cycle; included in the collection are the states with the greatest bit rates. Then the algorithm verifies that their anticipated completion time may satisfy the delay constraint. If not, add additional statuses — from high bitrate to low bitrate — not currently in the list. The weighted average of the bit-rate in the set F—obtained by adding up all status bit rates and dividing them by the set's associated equilibrium probability, must be calculated to estimate the algorithms' finish time, or t_f:

$$B_{avg} = \sum_{i \in F} \pi[i] * Br[i] \tag{4.24}$$

With the B_{avg}, we can calculate the expected finish time t_f according to Current Channel Status X_t if in the set:

$t_f = t + \frac{S_r}{B_{avg}} + T_{rest}$ if the Current Channel Status X_t is not in the set, the current time slot will not transmit the data.

$t_f = t + \frac{S_r - Br(X_t)}{B_{avg}} + T_{rest}$ if the Current Channel Status X_t is in the set, the current time slot will transmit the data.

The T_{rest} is the sum of the time used on the task execution by the cloud server T_{exec} and the time used on receiving the result from the cloud server T_{down}.

We must also confirm that they expect the finish time to satisfy the delay limitation when adding a new status to set F until the delay constraint is met or all status are in set F. After making progress, it will determine whether the anticipated completion time can be met under the given conditions. If not, see if the status X_t is included in the set; otherwise, move on to the following time slots. If not, determine the predicted energy use and decide whether to offload.

Water-Filling with Exponentiation (WF-Exp)

The OPO is also the foundation of this algorithm. However, in contrast to other algorithms, this one uses the transit matrix to track the progress of filling the water reservoir (Teymoori *et al.*, 2021). As a result, it is more accurate and performs better in terms of energy consumption. The WF-Equ is anticipated to make the most progress. Instead of doing this, though, use the equilibrium probability to estimate the expected finish time, t_f. It directly utilizes the channel status' transition probability.

Algorithm 5 Local execution starting time t_L, local execution energy E_L, job deadline t_D, transition probability matrix P, state space of the Markov chain $\mathcal{M} = \{1, 2, \ldots, m\}$, current state X_t, remaining size $S_r = S$.

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job size S.

1: **for all** $t = 1, \ldots, t_D$ **do**

2: **if** $S_r = 0$ **then**

break
end if
for all $i = 1, \ldots, m$ **do**
if $i == X_t$ **then**
$\alpha[i] = 1$
else
$\alpha[i] = 0$
end if
end for
$\mathcal{F} = \{m\}$ $\triangleright$ m is the state which has highest bitrate, at start, it only has one item.
if $X_t \notin \mathcal{F}$ **then**
Find minimum t_f which satisfied the equation $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
else
Find minimum t_f which satisfied the equation $S_r - Br(X_t) - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
end if
while $(t_f > t_D)$ and $(\mathcal{M} \neq \mathcal{F})$ **do**
choose the state with the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ and add to $\mathcal{F}$
if $X_t \notin \mathcal{F}$ **then**
Find minimum t_f which satisfied the equation $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
else
Find minimum t_f which satisfied the equation $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot$

$Br[i] \leq 0$

end if

end while

if $t_f > t_D$ **then**

break

end if

if $X_t \notin \mathcal{F}$ **then**

continue ▷ Continue with next timeslot $t+1$

end if

$E_{up} = \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot E_{tr}$

$E_l = \left(\max\left(t_f + 1, t_L\right) - \max\left(t, t_L\right)\right) E_L / T_L$

if $t_f \leq t_D$ **then**

$E_{off} = E_{up} + E_l + T_{down} E_{rc}$

else

$E_{off} = \dfrac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$

end if

if $E_{off} < \dfrac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ **then**

Offload at t

$S_r = \max\{0, S_r - Br\left(X_t\right)\}$

end if

end for

Each time slot should be lit up by the for-loop. Assumed to be M is the potential channel status set. The method will determine whether the remaining size is zero and continue iterating if it is not. Then it will create an array of the same size as the set M and mark the current state by changing $\alpha\left[X_t\right]$ to one while leaving the other elements at zero.

Start each iteration on time slots with an empty set F, much like the previous algorithms. It adds the status to the set in accordance with the corresponding bit rate, going from high to low, and calculates the anticipated completion time (t_f), which is held constant until all status have been added to the set or the t_f may meet the specified delay restriction. The method will stop if the delay limitation is not satisfied, but all status have been added to the set. To determine the t_f, locate the minimum value that is satisfied. When the set's current state X_t is empty:

$$S_r - \sum_{k=t+1}^{t_f} \sum_{i \in F} \left(\alpha P^{k-t}\right)[i] * Br[i] \leq 0 \tag{4.25}$$

If current status X_t is in the set:

$$S_r - Br(Xt) - \sum_{k=t+1}^{t_f} \sum_{i \in F} \left(\alpha P^{k-t}\right)[i] * Br[i] \leq 0 \tag{4.26}$$

After making progress, it will determine whether or not the anticipated completion time will satisfy the deadline limitation. If not, see if status X_t is included in the set; if not, move on to the following timeslots. If not, determine the predicted energy use and decide whether to unload.

4.4 Conclustion

In this chapter, we were given the pseudo code for the algorithms and explained how the algorithms make the offloading decision. We also do some analysis and comparison between the algorithms approached.

Chapter 5

Mobile Computation Offloading Over General Finite-State Markovian Wireless Channel

Besides the analysis on the GE model, we also study the energy-effective MCC over the multistate scenario. We also presented a dynamic offloading mechanism that can adjust data transmission rate optimally under the stochastic wireless channel environment. It is referred to as Online Optimal Scheduler (OnOpt-Sch). Like the OnOpt (Hekmati *et al.*, 2019a) aforementioned, it is an online algorithm to dynamically schedule an optimal number of bits to be transmitted in each time slot base on Dynamic Programming (DP).

We also applied Model reduction for Markov chains in the study. Because the large Markov Chains limited their application in practice (Hekmati *et al.*, 2019a, 2020; Teymoori *et al.*, 2021), we mitigated it by aggregating the state space of the underlying Markov chain into a smaller set of state. The approach is to build a decreased order model by grouping the states with strong interactions. The strongly interacting states within each group are known

as a super-node, and a Markov model that describes the transitions from one super-node to another are given(Phillips and Kokotovic, 1981; Yin and Zhang, 2012).

According to the paper (Zafer and Modiano, 2007), for a wireless fading channel with gain g, the energy consumed to transfer s bits of data is determined by the following convex monomial function given by:

$$e(s, g) = \mu \frac{s^n}{g} \tag{5.1}$$

In the equation, the n denotes the monomial order, and μ denotes the energy coefficient. In this study, we assumed $2 \leq n \leq 5$. This equation is used in many papers for bit-energy relationship (Neely *et al.*, 2003).

When we execute the jobs remotely, the data transmission energy consumption can be minimized by optimally varying the data rate. The relationship between the energy cost per time slot and transmitted bits per time slot is a convex function. To minimize the energy consumption, we need to transmit as few bits as possible(Uysal-Biyikoglu *et al.*, 2002). But on the other hand, reducing the data rate of the channel increases the total time consumption. An optimal data transmission rate schedule to minimize the total transmission energy as well as satisfy the deadline constraint is given by solving the problem:

$$\min_{\gamma \in \Gamma} \quad \mathbb{E}\{E_{off}(S, T, \gamma)\} \tag{5.2}$$

$$s.t. \quad \sum_{t=1}^{T} s_t = S$$

$$s_t \geq 0.$$

The $\gamma = \{s_1, s_2, ...s_T\}$ denotes a data transmission schedule that meets the delayed deadline (T time slots), Γ is the set of all feasible data scheduling vectors, the expected value $\mathbb{E}\{E_{off}(S, T, \gamma)\}$ is known as an estimation of the transmission energy.

Under the optimal transmission scheduling γ^*, the minimum amount of the expected transmission energy for the cloud execution is denoted by:

$$E^*_{off} = \mathbb{E}\{E_{off}(S, T, \gamma^*)\}. \tag{5.3}$$

Then, the mobile device can make an optimal offloading decision, $a \in \{0, 1\}$, $a = 0$ and 1 indicate local and remote execution, respectively according to:

$$a = \begin{cases} 0, & E_L \leq E^*_{off} \\ 1, & E_L > E^*_{off} \end{cases} \tag{5.4}$$

5.1 Optimal Solution Using DP Approach

Assume that t is the index of time in descending order (from T to 1), and l_t denotes the number of unfinished bits at time slot t, then the optimal transmit data size scheduling vector is given by:

$$s^*_t(l_t, g_t, G') = \begin{cases} l_t \left[\frac{(g_t)^{\frac{1}{n-1}}}{(g_t)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G_0}})^{\frac{1}{n-1}}} \right], & t \geq 2 \\ l_1, & t = 1. \end{cases} \tag{5.5}$$

The G' denotes the initial channel state and can be any value of the state space $\mathcal{G}$, and Z_t is calculated recursively according to equation (5.6).

$$Z_t; G' = \begin{cases} \sum_{G_i \in \mathcal{G}} P_{G'G_i} \left[\left(\frac{1}{(G_i)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G'}})^{\frac{1}{n-1}}} \right)^{n-1} \right], & t \geq 2 \\ \sum_{G_i \in \mathcal{G}} P_{G'G_i} \left[\frac{1}{G_i} \right], & t = 1 \end{cases} \tag{5.6}$$

And the minimum expects energy consumption:

$$E^*_{off}(L, T; G') = \mu L^n Z_{T,G'}, \tag{5.7}$$

and therefore, the minimum expected transmission energy could be found using (5.8).

$$E^*_{off}(L, T) = \sum_{G_i \in \mathcal{G}} \frac{T_{G_i}}{\sum_{G_i \in \mathcal{G}} T_{G_i}} E^*_{off}(L, T; G_i) \tag{5.8}$$

Like the OnOpt, we use the DP to speed up our computing process, the optimization problem given by:

$$J_t(l_t, g_t) = \begin{cases} \min\limits_{0 \leq s_t \leq l_t} \left(\mu \frac{{s_t}^n}{g_t} + \mathbb{E}\left(J_{t-1}(l_t - s_t, g_{t-1}) \right) \right), & t \geq 2 \\ \mu \frac{{s_1}^n}{g_1}, & t = 1. \end{cases} \tag{5.9}$$

Here, l_t denotes the number of remaining (un-transmitted) bits at t, with $l_{t-1} = l_t - s_t$. We use induction to prove (5.6). At $t = 1$, all the remaining l_1 bits have to be transmitted to meet the deadline constraint. Without loss of generality, by assuming that the channel at $t = 2$ is in the state $g_2 = G'$, the expected minimum energy is given by:

$$\bar{J}_1(l_1, g_1; G') = \mathbb{E}\left[\mu \frac{{l_1}^n}{g_1} \right] = \mu {l_1}^n \sum_{g_i \in \mathcal{G}} p_{G'G_i} \left[\frac{1}{G_i} \right] = \mu {l_1}^n Z_1 \tag{5.10}$$

Suppose (5.10) is valid for $t-1$. Therefore,

$$\bar{J}_{t-1}(l_t - s_t, g_{t-1}; G') = \mu(l_t - s_t)^n Z_{t-1} \tag{5.11}$$

Now, the DP problem stated in (5.11) becomes

$$J_t(l_t, g_t; G') = \min_{0 \le s_t \le l_t} \left(\mu \frac{{s_t}^n}{g_t} + \mu(l_t - s_t)^n Z_{t-1;G'} \right) \tag{5.12}$$

The optimal s_t, denoted as s_t^* , can be solved as:

$$s_t^*(l_t, g_t; G') = \frac{l_t g_t{}^{\frac{1}{n-1}}}{(g_t)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G'}})^{\frac{1}{n-1}}} \tag{5.13}$$

By substituting (5.13) to (5.12), we have:

$$J_t(l_t, g_t; G') = \mu l_t^n \left[\left(\frac{1}{(g_t)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G'}})^{\frac{1}{n-1}}} \right)^{n-1} \right] \tag{5.14}$$

By taking expectation of $J_t(l_t, g_t; G')$ with respect to g_t and after simplification, we have:

$$Z_{t;G'} = \mathbb{E} \left[\left(\frac{1}{(g_t)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G'}})^{\frac{1}{n-1}}} \right)^{n-1} \right] \tag{5.15}$$

$$= \sum_{G_i \in \mathcal{G}} p_{G'G_i} \left[\left(\frac{1}{(g_t)^{\frac{1}{n-1}} + (\frac{1}{Z_{t-1;G'}})^{\frac{1}{n-1}}} \right)^{n-1} \right]$$

5.2 Approximate Solution Using Markovian Aggregation

As aforementioned, at each single time slot, the online algorithm recursively solves a separate DP with updated input parameters to find the optimal number of bits to be transmitted at each time

slot. The computational complexity of these DP problems is very large if the Markov chain is very large. This has a serious impact on the execution time of the algorithm, and it can not use in online mobile implementation. A widely used approach to reducing the state space of a Markov chain is by aggregating groups of nodes in the original transition graph with the help of partitioning functions.

The distance between the Markov model and its approximation:

$$R(P||Q) = \sum_{i,j \in \mathcal{N}} \pi_{i,j} P_{i,j} log\left(\frac{P_{i,j}}{Q_{i,j}}\right) \tag{5.16}$$

The optimal transitioning probability matrix of the lifted Markov chain, denoted as $\mathcal{Q}'$, for a given partitioning function can be obtained by:

$$\mathcal{Q}'_{ij}(\phi) = \frac{\pi_j}{\sum_{k \in \psi(j)} \pi_k} Q_{\phi(i)\phi(j)} \tag{5.17}$$

Where, π is the stationary probability distribution on the state space $\mathcal{N}$, and $\psi(j)$ denotes the set of states belonging to the same super-node as the j^{th} state.

The KLDR is given by:

$$R_\phi(P||\mathcal{Q}) := R(P||\mathcal{Q}'(\phi)). \tag{5.18}$$

And we need to solve the problem:

$$\min_{\phi,Q} \quad R_{\phi,Q}(P||Q)\} \tag{5.19}$$

$$s.t. \quad \sum_{l \in \mathcal{M}} Q_{kl} = 1, \quad k \in \mathcal{M} \tag{5.20}$$

$$Q_{kl} \geq 0, \quad k, l \in \mathcal{M}. \tag{5.21}$$

the optimal Q matrix and its invariant distribution can be easily obtained according to:

$$Q_{kl}(\phi) = \frac{\sum_{i\in\phi^{-1}(k)} \sum_{j\in\phi^{-1}(l)} \pi_i P_{ij}}{\sum_{i\in\phi^{-1}(k)} \pi_i} \tag{5.22}$$

$$\theta_k(\phi) = \sum_{i\in\phi^{-1}(k)} \pi_i, \quad k \in \mathcal{M}. \tag{5.23}$$

Reduces the aggregation problem to the minimization of the K–L divergence $R_\phi(P||Q(\phi))$ rate over ϕ, which can be written as:

$$R_\phi(P||Q(\phi)) = I(P) - I(Q(\phi)). \tag{5.24}$$

Where, the function $I(X)$, represents the mutual information of the matrix X, which is the difference between entropy the Markov chain and the entropy of its invariant distribution. Therefore, the optimization problem (5.19) is given by:

$$\max_{\phi:\mathcal{N}\to\mathcal{M}} \quad I(Q(\phi)). \tag{5.25}$$

The combinatorial optimization problem mentioned upon can be approximated by the following continuous nonlinear program:

$$\max_{\nu\in\mathbb{R}^n} \quad F(\nu) \tag{5.26}$$

Assume that ν is an n by 1 real-valued vector. The relation between ϕ and ν is given by:

$$\phi(i) = \begin{cases} 1, & v(i) > 1 \\ \\ 2, & v(i) \leq 1 \end{cases} \tag{5.27}$$

The matrix $Q(\nu)$, which is an approximation of $Q(\phi)$, can be expressed as:

$$Q(\nu) = \begin{bmatrix} \frac{\alpha(\nu)}{\beta(\nu)} & \frac{\beta(\nu)-\alpha(\nu)}{\beta(\nu)} \\ \frac{\beta(\nu)-\alpha(\nu)}{1-\beta(\nu)} & \frac{1-2\beta(\nu)+\alpha(\nu)}{1-\beta(\nu)} \end{bmatrix}. \tag{5.28}$$

Where, $\alpha(\nu) = \nu'\Pi P\nu$, $\beta(\nu) = \nu'\Pi\nu$, and Π=diag(π). Followed from (5.23), the invariant distribution $Q(\nu)$ is $\theta(\nu) = [\beta(\nu), 1-\beta(\nu)]$.

Since $Q(\nu)$ and $\theta(\nu)$ can be represented in terms of $\alpha(\nu)$ and $\beta(\nu)$, the objective function $F(\nu)$ can be expressed as:

$$F(\nu) = I(Q(\nu)) = h(\theta(\nu)) - H(Q(\nu)) = \hat{F}(\alpha(\nu), \beta(\nu)). \tag{5.29}$$

In order to get the solution for the relaxation problem (5.25), assume the critical points of the function in $\mathbb{R}^n$ by setting the derivative to zero. Solving $\frac{\partial F}{\partial \nu} = 0$ results in equations (5.30) and (5.31)

$$\hat{P}\nu = \lambda\nu \tag{5.30}$$

$$\alpha(\nu) = \beta(\nu)^2 \tag{5.31}$$

Where, $\hat{P} = \frac{1}{2}(P + \Pi^{-1}P'\Pi)$, and

$$\lambda = \frac{log\frac{(1-\beta)^2(\beta-\alpha)^2}{\beta^2(1-2\beta+\alpha)^2}}{log\frac{(\beta-\alpha)^2}{\alpha(1-2\beta+\alpha)}}. \tag{5.32}$$

if ν is a spectral critical point, then

$$\alpha(\nu) = \frac{\nu'(\Pi P + P'\Pi)\nu}{2} = \nu'\Pi\hat{P}\nu = \lambda\nu'\Pi\nu = \lambda\beta(\nu). \tag{5.33}$$

At such spectral critical points, both the function F and the matrix Q can be expressed as a

function of the eigenvalue $\lambda \in \{\lambda_2, \cdots, \lambda_n\}$ according to what follows:

$$F(\nu) = \hat{F}(\lambda B(\lambda), B(\lambda)) = F^*(\lambda) \tag{5.34}$$

$$Q(\nu) = \begin{bmatrix} \lambda & 1-\lambda \\ \frac{B(\lambda)-\lambda B(\lambda)}{1-B(\lambda)} & \frac{1-2B(\lambda)+\lambda B(\lambda)}{1-B(\lambda)} \end{bmatrix}. \tag{5.35}$$

From the resulting n possible choices of (n+1)-partitions, we select the one that minimizes $R_{\phi_{n+1}^{(i)}}(P||Q(\phi^{(i)}))$ according to:

$$i_{min} =_{i\in\{1,2,\ldots,n\}} R_{\phi_{n+1}^{(i)}}(P||Q(\phi_{n+1}^{(i)})) \tag{5.36}$$

Then, the associated reduced order Markov model is given by $Q(\phi^{(i_{min})})$. The recursive algorithm continues until the predetermined threshold value ε for the modeling error is reached, i.e., the algorithm is terminated if:

$$|R_{\phi_m^{(i_{min})}}(P||Q(\phi_m^{(i_{min})})) - R_{\phi_{m-1}^{(i_{min})}}(P||Q(\phi_{m-1}^{(i_{min})}))| < \varepsilon \tag{5.37}$$

Chapter 6

Performance Evaluation

6.1 Introduction

In this study, we conduct trials on many elements and contrast the effectiveness of all methods we mentioned. In our experiments, the GE model is used for every wireless channel. Details of the channel model and aspects of the algorithms have already been covered in earlier chapters.

6.2 Mobile Computation Offloading Over Gilbert-Elliot channel

The Comparative Study On The Energy Consumption And The Different Time Constraints With The Same Data Size

In these studies, we compare the performance of various offloading decision algorithms when performing a task that requires little computational power and median data size in a poor wireless environment under various delay constraints. For the entire assignment, we set the data size S_{up} to

$700KB$. Additionally, the task requires $30M$ CPU cycles. The size of each time slot is $1ms$. The r_G and r_B broadcast at a rate of $5k$ per time slot and $30k$ per time slot, respectively. The frequency of the mobile device's CPU is $1000Mhz$, which translates to 1 M CPU cycles every time slot. The mobile device uses $2W$ or $2mj$ of electricity when performing tasks locally. The E_L is thus $60mj$. In this case, the $P_{GG} = P_{GB} = P_{BB} = P_{BG} = 0.5$ for the channel state. We equally divided the task for the LARAC algorithms to execute in the joint task. We discovered that the quantity of subtasks greatly impacts energy usage. Therefore, in our studies, we assess the LARAC algorithms' performance in two scenarios. In the first instance, we divided the task into five equally sized sections, with each sub-upload task and download sizes totalling $\{$140,140,140,140,140$\}$. There are $\{$6,6,6,6,6$\}$ units of work. The task was split into 10 equally sized subtasks in the second scenario. Subtasks have an upload size of $\{$70,70,70,70,70 ,70,70,70,70,70$\}$ and a download size of $\{$70,70,70,70,70 ,70,70,70,70,70$\}$ with a workload of $\{$3,3,3,3,3,3,3,3,3,3$\}$. The range of deadlines is [30,120]. In this scenario, each algorithm will be executed 1000 times, calculating the average value for each configuration.

The graph depicts the algorithms' energy usage as the job deadline t_D approaches. We discovered that most soft-deadline constraint algorithms perform better on energy consumption; they need not complete the task concurrently to ensure it is finished by the deadline. The algorithms for the hard-deadline restriction are chosen to perform the task on the local devices since it is difficult to satisfy the hard deadline with minimal t_D. The deadline is easier to meet, and the algorithms are more likely to offload the task to the remote server as the t_D increases. The likelihood of finding a shorter time window to complete the offloading, which would reduce energy usage, is likewise increased by an increase in t_D. This also applied to all algorithms in our tests. The algorithms can almost always discover successive time slots with a satisfactory status when the t_D is large enough, so continuing to raise the t_D will not lower the energy usage on a mobile device. The collaborative job execution employing the LARAC algorithms with 10 parts provides the best performance on energy usage, according to our algorithm performance analysis. It is because this method can

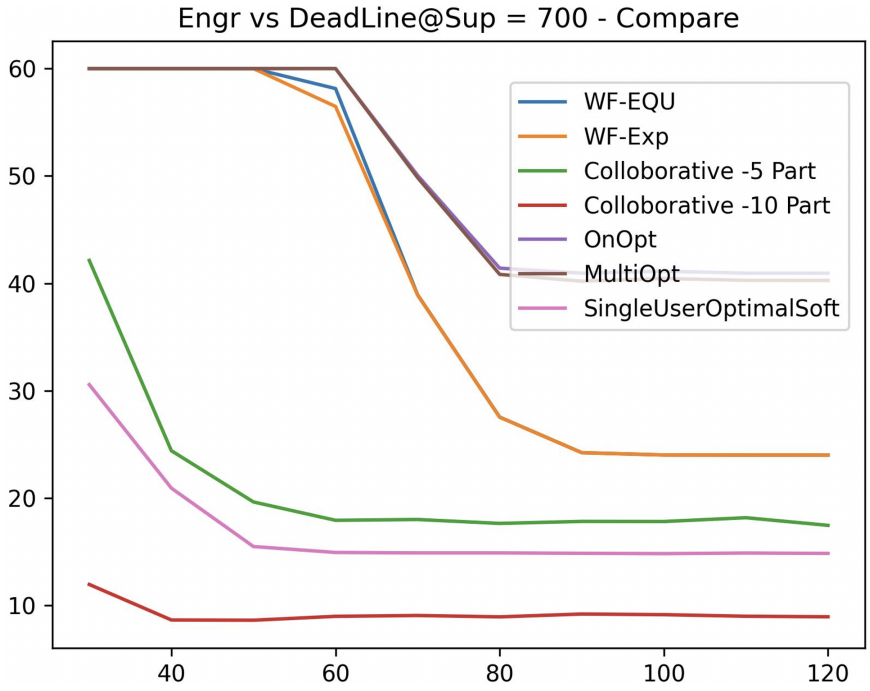

Figure 6.1: The Energy and the Deadline with the task size 700kb

offload the task into several sections and is a soft-deadline algorithm. Additionally, it is superior to the same five-part algorithms because adding more parts will make the work more flexible. Two LARAC setups offer the best single-user energy usage performance. It is also a soft-deadline constraint algorithm. MultiOpt, the 'multi-part version' of OnOpt, has superior performance on energy consumption since it is more flexible than OnOpt when comparing the algorithms for hard deadlines, such as the two configurations on the LARAC. The WF-EQU and WF-EXP are built on the preemptive offloading principle. They outperform the OnOpt and MultiOPT algorithms by a wide margin. Compared to the WF-EQU, the WF-EXP has no advantage regarding energy consumption. The decision-making process for the Markov chain channel model is more accurate when based on the transition probability rather than the equilibrium probability.

Now we change the task settings based on the first experiment. The CPU cycles required by the task increases to 40M CPU cycles, causing the increase of the local execution time. Hence, we

adjusted the range of deadline to [40, 130]. For collaborative offload, similar with the first experiment, we divided the task into 5 or 10 equally sized sections, leaving the other settings untouched. The graph below depicts the results:

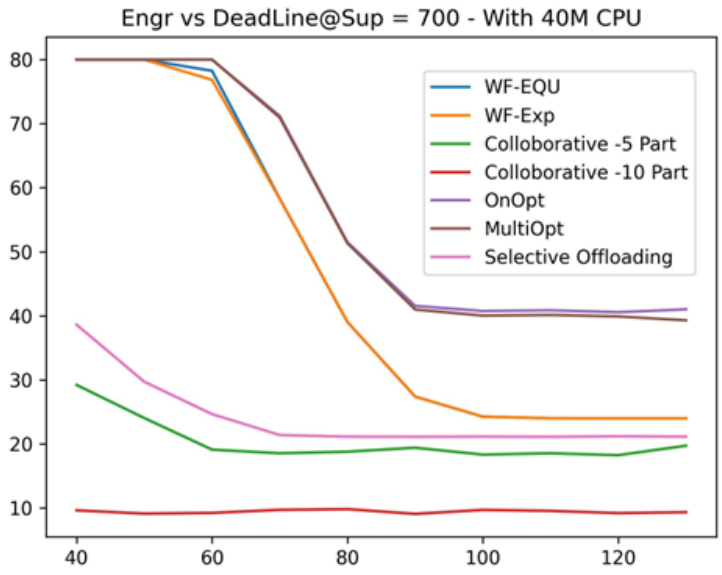

Figure 6.2: The Energy and the Deadline with the task size 700kb and 40M CPU cycles for task

From the results, we can discover that, besides the increased energy consumption, the performance of collaborative offloading employing LARAC algorithms did not change much, showing better performance than selective offloading in both experiments.

We then increased the S_{up} to 800. The task is divided into 5 or 10 equally sized sections and other settings stays the same as experiment one. The graph below is created to show the results of collaborative offloading:

From the result we could see that, other than the extra energy consumption caused by the increase of S_{up}, the performance ranking among different offloading algorithm did not change.

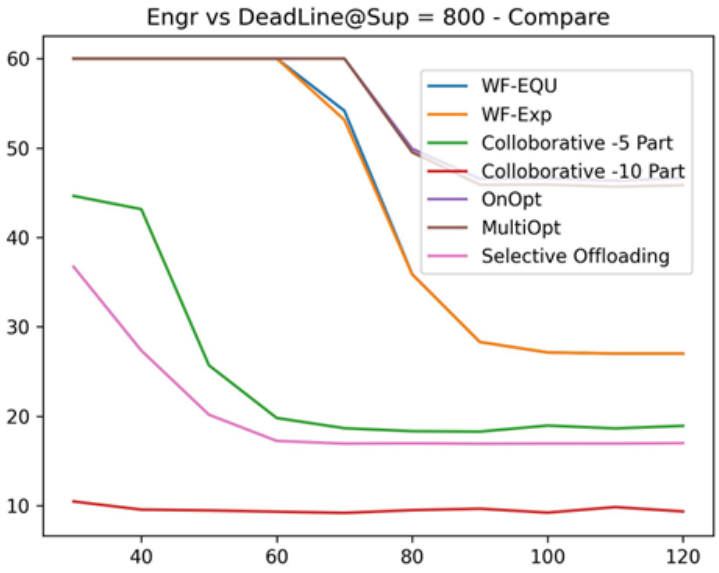

Figure 6.3: The Energy and the Deadline with the task size 800kb

The Comparative Study On The Energy Consumption And The Different Data Size With The Same Deadline

The data size S_{up} range for the entire task in these trials was set at [100,1500]. Also, the task requires $30M$ CPU cycles. A time slot has a length of 1 ms. The r_G and r_B transmit at a $40k$ per time slot and $4k$ per time slot, respectively. The mobile device's CPU operates at $1000Mhz$—or $1M$ CPU cycles per time slot. The power consumption of the mobile device when performing a task locally is $2W$ or $2mj$ per time slot. The E_L is thus $60mj$. In this case, the $P_{GG} = P_{GB} = P_{BB} = P_{BG} = 0.5$ for the channel state. We equally divided the task for the LARAC algorithms to execute in the joint task. We discovered that the number of subtasks greatly impacts energy usage. Therefore, in our studies, we assess the LARAC algorithms' performance in two scenarios. The upload size range for each of the five identical subtasks in the first scenario is [20,300], and the download size range is equally [20,300]. There are $\{6,6,6,6,6\}$ units of work. In the second instance, the task was broken into 10

equally sized subtasks. The range of upload and download sizes for each sub-task is [10,150], and the range of workloads is $\{3,3,3,3,3,3,3,3,3,3\}$. The time limits are $120ms$.

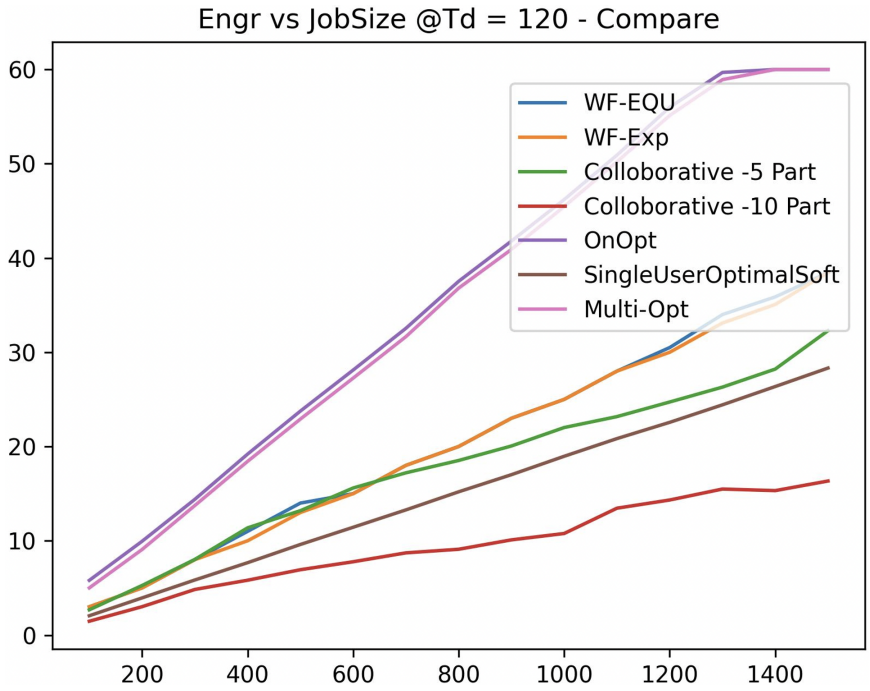

Figure 6.4: The energy and data size under the same deadline

The experiment graph demonstrates how much power the mobile device uses as the data size S rises. Thus, we discovered that most soft deadline algorithms perform better regarding energy consumption than the hard deadline algorithms since they do not need to complete the task concurrently to ensure its completion by the deadline.

All algorithms in our testing choose to offload the work to the distant server when S is small because offloading can probably meet the delay limitation without local execution. The energy consumed during the wireless transmission of the task between the mobile device and remote server increases as the task's data size S increases. Also, some algorithms, such as OnOpt and MultiOpt, may carry out the task on the local device due to the increased time and energy usage. Further, we discovered that the LARAC algorithms with 10 pieces perform best-regarding energy usage when used to execute collaborative tasks because this method can offload the task into several sections and

is a soft-deadline algorithm. Having the same algorithms divided into five sections is preferable because the task will be more adaptable if it is divided into multiple pieces. Two LARAC setups offer the best single-user energy usage performance and functions as a soft deadline constraint algorithm. Due to its greater flexibility compared to the OnOpt, the MultiOpt is the 'multi-part version' of the OnOpt and performs better when comparing algorithms for hard deadlines, such as the two setups on the LARAC. The WF-EQU and WF-EXP are built on the Preemptive Offloading principle, outperforming the OnOpt and MultiOPT algorithms by a wide margin. Compared to the WF-EQU, the WF-EXP has no energy consumption advantage. Decision-making for the Markov chain channel model is more accurate when based on the transition probability rather than the equilibrium probability.

6.3 Mobile Computation Offloading Over Finite-State Markovian Chain

In this experiment, we use a 6-state Markov chain (i.e., $\mathcal{G} = \{G_1, G_2, \ldots, G_6\}$) to model the communication channel. We assume that state G_1 represents the channel outage that may occur because of shadowing or other similar phenomena; for the other states, higher states have higher channel gains. The assigned channel gains to states G_1 to G_6 are 0, 0.2, 0.4, 0.6, 0.8, 1, respectively. To investigate the effect of being NDMC on the approximation methods, we conduct the simulations over the transitioning probability matrix (6.1) for $\epsilon = 0.05$ and 0.25. Note that as ϵ increases, the matrix gets farther from being NDMC.

$$\mathbb{P} = \begin{bmatrix} 0.5-\epsilon & 0.5 & 0.5 & \epsilon & 0 & 0 \\ 0.5 & 0.5-\epsilon & 0.5 & 0 & \epsilon & 0 \\ 0.5 & 0.5 & 0.5-\epsilon & 0 & 0 & \epsilon \\ \epsilon & 0 & 0 & 0.5-\epsilon & 0.5 & 0.5 \\ 0 & \epsilon & 0 & 0.5 & 0.5-\epsilon & 0.5 \\ 0 & 0 & \epsilon & 0.5 & 0.5 & 0.5-\epsilon \end{bmatrix} \tag{6.1}$$

The default parameters used in the simulations are given as follows. Each time slot is taken to be 1 ms. A task with computational load $W = 10$ M CPU cycles is considered. The task completion deadline T is set to 60-time slots. The local execution energy per CPU cycle is $e_l = 2 \times 10^{-6}$ mJ and the local computation speed is $f_l = 1$ M CPU cycles per time slot (Sardellitti *et al.*, 2015) (Chen *et al.*, 2015). Therefore, the local execution time is $T_L = \frac{D}{f_l} = 10$ time slots; the local energy consumption is $E_L = e_l D = 20$ mJ. We assume $E_{down} = T_{down} = T_{exec} = 0$ time slot. When collecting the simulation results, the energy consumption for running the online algorithm for the mobile device was assumed to be negligible compared to that for transmitting to the cloud server. This is a common assumption when the amount of data to upload the task is large. Despite this, we included graphs of the relative running times of the algorithms so this component could be included if required.

For comparison, we plot three other algorithms: *Local execution*, *Always Offloading*, and *Off-Opt-Sch*. The execution algorithm executes the entire job locally without any offloading. For the always offloading algorithm, offloading starts immediately at the task release time without any local execution.

The Off-Opt-Sch algorithm is an offline lower bound on energy—derived by solving an offline version of the problem where the communication channel is considered deterministic. Simulations are also conducted by applying the proposed Markovian aggregation method to approximate the online-optimal algorithm's solution. The resulting algorithm is referred to as *On-Approx-sch*.

Figure 6.5 shows the average energy consumption of the mobile device as the job size S changes. In this set of simulations, ϵ equals 0.05. As seen from the figure, when S is small, the offloading algorithms may decide to upload as few bits as possible at each time slot, resulting in lower overall energy consumption. In this case, the average energy consumption of the offloading algorithms is close to the lower bound Off-Opt-Sch. As S increases, the algorithms try to schedule as many bits as possible in the earlier time slots to ensure the deadline is met, which will lead to an increase in energy cost, so there will be more gap between the online algorithms and the offline bound. When

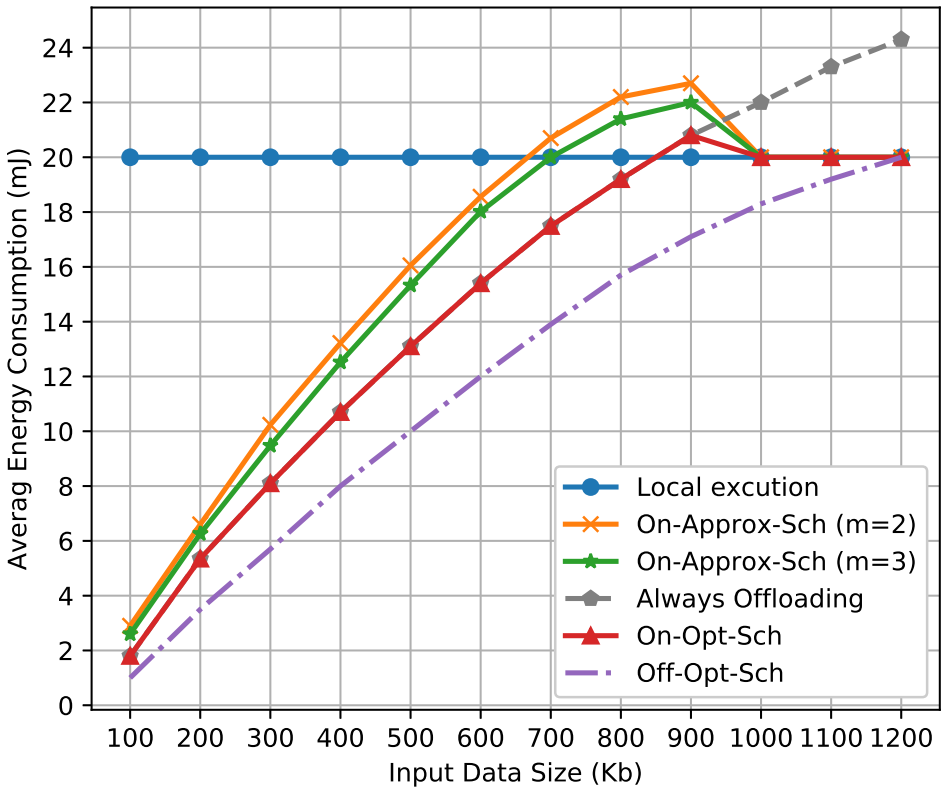

Figure 6.5: Average energy cost vs. the input data size S: $\epsilon = 0.05$.

S is sufficiently large, all the offloading algorithms decide not to offload, resulting in the same energy consumption as in Local Execution . By increasing the number of super-states (i.e., m), the aggregated chain becomes a more precise approximation of the original, resulting in more accurate offloading decisions, reducing the mobile device's average energy consumption. However, since the original Markov chain for $\epsilon = 0.05$ is an NDMC, increasing the number of partitions has little impact on the approximation error.

Figure 6.6 illustrates the average energy cost of the mobile device versus the input data size S when ϵ is set to 0.25. By comparing Figures 6.6 and 6.5, we find that increasing the value of ϵ results in approximately the same average energy consumption for all algorithms except On-Opt-Sch algorithms because, in this case, the original Markov chain is less decomposable than the previous

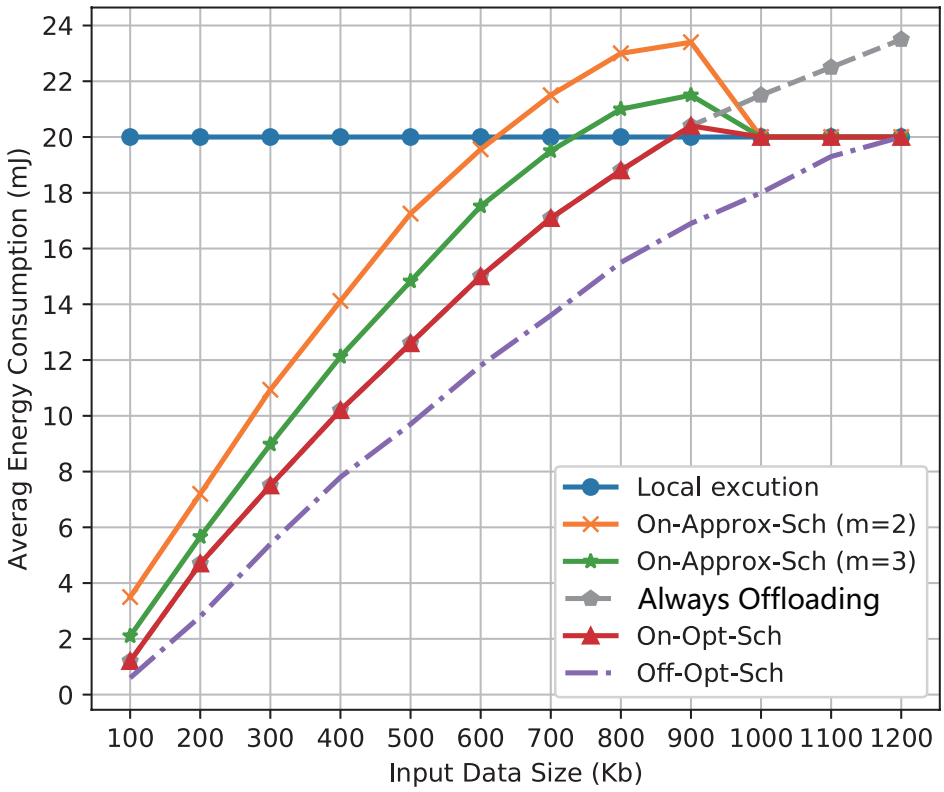

Figure 6.6: Average energy cost vs. the input data size S: $\epsilon = 0.25$.

case. Therefore, increasing the number of partitions reduces the approximation error.

Figure 6.7 indicates the average running time of the algorithms as the input data size S varies. As seen from this figure, the running time of the approximation-based algorithms is much less than that of the the optimal online algorithm, and the gaps between the running times increase as the job size enlarges.

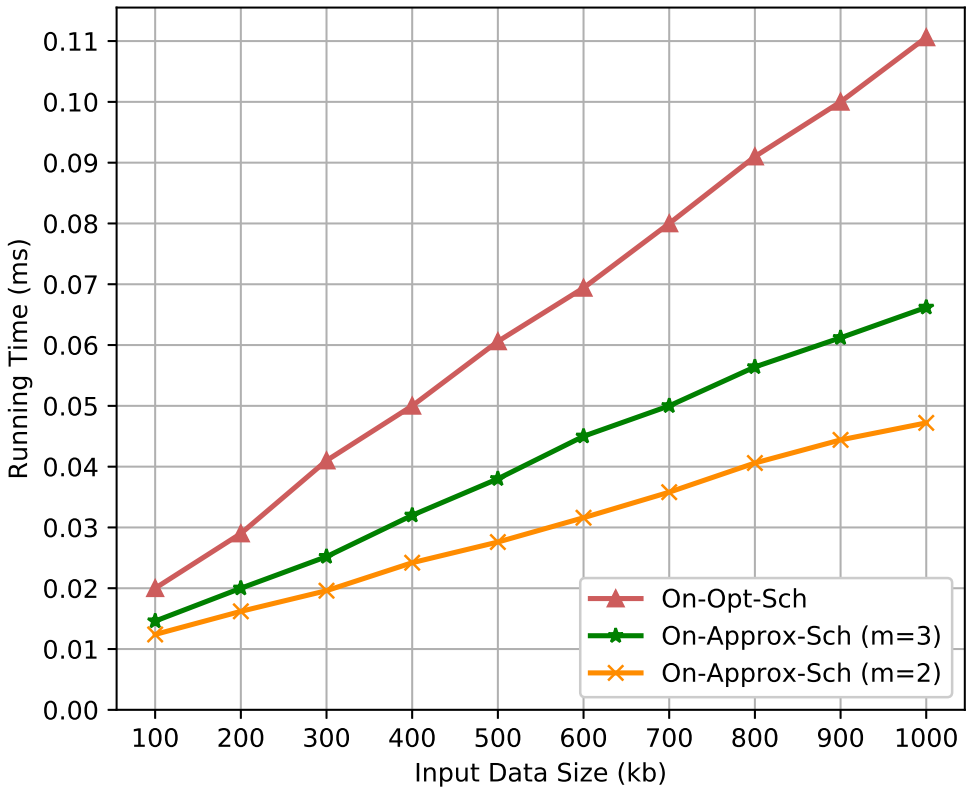

Figure 6.7: Running time vs. the input data size.

Chapter 7

Conclusion & Future Work

In this thesis, we examine numerous state-of-the-art papers covering the Mobile Cloud Computing (MCC) and the Mobile Computation Offloading (MCO) from various angles. We examine the market's desire for MCCs and their value to the market, including how the MCO has benefited mobile computing, the network status and development trends of mobile devices. From the references, we know that the MCC and MCO can extend the battery life and overcome resources constraint of the mobile devices. Still, MCO can also replace the remote desktop/VDI in some scenarios and let the user obtain a more fluid experience. We compare the various types of network topology used in the MCC from many articles (Yang *et al.*, 2013, 2019a; Zhang *et al.*, 2017a; Habak *et al.*, 2015; Chhadva *et al.*, 2021; Wang *et al.*, 2016).

We analyzed sample data to clarify the connection between the MC, MCC, and cloud computing. Given a general understanding of cloud computing, MC, MCC, and MCO, we also consider the challenges faced by MCC and MCO, such as the application partition and its heterogeneous execution environment. We also reviewed the relevant literature to learn more about the various existing frameworks' approaches and designs. After reading various academic works, we decided to base our thesis on comparing single-user offloading techniques using stochastic wireless transmission channels. To conduct our experiments, we chose 6 algorithms.

Our trials compare the algorithms under the soft-deadline and hard-deadline conditions. The biggest distinction is that the deadline is always met in a situation with a hard deadline. The concurrent local execution (CLE) has been suggested in the algorithms dealing with the hard-deadline case. This enables the work to be executed simultaneously on a local device and a remote server and is used to guarantee the task's completion before the deadline in the case of a random channel. The CLE method can ensure the deadline is always met, even in worst-case scenarios, such as a wireless network interruption during offloading.

The algorithms selected in this experiment also include the continuing and multipart offloading approaches for offloading. The wireless channel model used in our investigations is modeled as a Markov chain. More precisely, we employ the Gilbert-Elliott Model (GE), a two-state Markov chain model that only has 'Bad' and 'Good' states. The transition probability matrix determines the changes between states of the various bitrates in each time slot. During the offload, the mobile device is aware of the transition probability matrix. We research and evaluate the effectiveness of various MCO offloading decision-making algorithms.

The details of our channel model and various offloading models for the soft and hard deadline constraints are in Chapter 3, which explains how to build the TDAMC model utilizing the time series and Markov chain for the experiments. In Chapter 3, we also describe how the GE model and Markov chain are related using our channel concept as an example. Then we discuss four specific scenarios: hard deadlines with continuous offloading, multipart offloading with a hard deadline, continuous offloading with a soft deadline, and multipart offloading with a soft deadline. We compare their differences regarding the deadline and the offloading strategy, considering the definition of the energy model for each circumstance.

We review the approaches of the six algorithms in Chapter 4, including the justification for the equation and algorithms. We explain the guidelines for adjusting the CPU for local execution and the transfer schedule for single-user optimal soft-deadline algorithms. We describe how the LARAC translates a problem of task execution into a graph shortest path problem in the section

about the LARAC algorithms in collaborative task execution. Using the DP in OnOpt and MultiOpt to minimize the algorithm's time complexity, as well as the relationship and distinction between the OnOpt and MultiOpt, are likewise covered in the same way for the other four algorithms facing a hard deadline. The WF-EXP and WF-WQU designs, which use the greedy algorithm, tackle the OPO issue with much less time complexity. Much more, while the algorithms aim to decrease the energy consumption on MCO while satisfying the delay constraint, some algorithms are approached by using a strategy to adjust the device's CPU and transfer speed frequently. In contrast, other algorithms divide the task into multiple parts and choose where each part will be executed. Choosing when to begin uploading data to a remote server is one way to implement it.

The graph showing the experiment outcome for each of the six algorithms we chose in various scenarios is in Chapter 5, including the energy consumption concerning the deadline in the same wireless channel environment and the energy consumption concerning the amount of data for offloading. We compare how well the algorithms performed regarding energy usage in the same simulated situations with the same configuration. We analyze the experiment's findings. We learn from the trials that the soft deadline is typically more energy-efficient, suggesting that choosing the deadline limitations for the task in accordance with demand also contributes to lowering energy usage in the real world. The multipart offloading algorithm performs better in energy consumption for both soft and hard deadline constraints. Two OPO-based algorithms perform better than the other two algorithms in the settings with hard deadlines.

Future projects: In most scenarios, a base station or a access point need to provide the service simultaneously. However, our study only focuses on the study of single-user scenarios. In real settings, multiple users are multitasking in the network and may interfere with each other, like the interference caused by noise in wireless channels. In the future, we need to further study the performance of different algorithms in multi-user scenarios.

The fast advancement of machine learning, such as offloading based on deep learning models, also provides new opportunities for Mobile Cloud Offloading (MCO).

The usage of Mobile Cloud Computing (MCC) is not only limited to mobile devices. Vehicular Edge Computing(VEC) and Internet of Things (IoT) can also leverage MCC, so further research on these new use cases are required. The needs of VEC and IOT are different from those of smart mobile devices (SMD) we commonly use. VEC travels at a much faster speed than common SMD, and IoT has totally different mechanisms for information transfer and battery capacity.

The performance tuning of MCC and MCO targeting on varying traffic volume, different applications and different devices is noteworthy. Since the offloading algorithms consume time and computing resources, to improve time complexity of the algorithms and optimize the resource consumption of offloading is another area for further study. Additionally, offloading algorithms performs differently in different types of networks, such as WiFi or mobile network, which is worth researching.

During the MCC, a large volume of data, which possibly include personal or sensitive data of users, is transferred to remote servers, imposing concerns on information security. Secured data transfer is required to process user data, The user data should be transferred via secured connections or encoding protocols. The impact of these approaches on the performance of data transfer should be gauged in future search.

The development of mobile applications which are compatible with MCC is crucial to promote the usage and build an ecosystem of MCC. How to enhance the existing applications with MCC functionality and allow developers to create MCC compatible applications with ease using frameworks such as MAUI is a prominent topic for future research.

Bibliography

Abolfazli, S., Sanaei, Z., Alizadeh, M., Gani, A., and Xia, F. (2014a). An experimental analysis on cloud-based mobile augmentation in mobile cloud computing. *IEEE Transactions on Consumer Electronics*, **60**(1), 146–154.

Abolfazli, S., Sanaei, Z., Gani, A., Xia, F., and Lin, W.-M. (2014b). Rmcc: Restful mobile cloud computing framework for exploiting adjacent service-based mobile cloudlets. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 793–798. IEEE.

Ahmadi, M., Khanezaei, N., Manavi, S., Moghaddam, F. F., and Khodadadi, T. (2014). A comparative study of time management and energy consumption in mobile cloud computing. In *2014 IEEE 5th Control and System Graduate Research Colloquium*, pages 199–203. IEEE.

Aldahwan, N. S. and Saleh, M. S. (2018). Developing a framework for cost-benefit analysis of cloud computing adoption by higher education institutions in saudi arabia. In *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, pages 1–9. IEEE.

Bangui, H., Rakrak, S., and Raghay, S. (2015). External sources for mobile computing: The state-of-the-art, challenges, and future research. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, pages 1–8. IEEE.

Cau, E., Corici, M., Bellavista, P., Foschini, L., Carella, G., Edmonds, A., and Bohnert, T. M. (2016). Efficient exploitation of mobile edge computing for virtualized 5g in epc architectures.

In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pages 100–109. IEEE.

Chen, X., Jiao, L., Li, W., and Fu, X. (2015). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, **24**(5), 2795–2808.

Chhadva, Y., Agrawal, R., Badlani, S., and Devadkar, K. (2021). Architecture for mobile cloud computing using five level offloading (armflora). In *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1280–1285. IEEE.

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

Dehnie, S. (2007). Markov chain approximation of rayeleigh fading channel. *IEEE International Conference on Signal Processing and Communications*, pages 1311–1314.

Farrugia, S. (2016). Mobile cloud computing techniques for extending computation and resources in mobile devices. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 1–10. IEEE.

Forman, G. H. and Zahorjan, J. (1994). The challenges of mobile computing. *Computer*, **27**(4), 38–47.

Futurenetworks.ieee.org (2022). Ieee future networks: Enabling 5g and beyond. `https://futurenetworks.ieee.org/images/files/pdf/ieee-5g-roadmapwhite-paper.pdf`.

Grand View Research, I. (2022). 5g services market size worth $1.67 trillion by 2030 with cagr of 52.0 `https://www.prnewswire.com/news-releases/5g-services-market-size-worth-1-67-trillion-by-2030-with-cagr-of-52-0-grand-vie`

w-research-inc-301503805.html#:~:text=5G%20Infrastructure%20Market%20%2D%20The%20global,49.8%25%20from%202021%20to%202028.

Habak, K., Ammar, M., Harras, K. A., and Zegura, E. (2015). Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In *2015 IEEE 8th international conference on cloud computing*, pages 9–16. IEEE.

Hekmati, A., Teymoori, P., Todd, T. D., Zhao, D., and Karakostas, G. (2019a). Optimal mobile computation offloading with hard deadline constraints. *IEEE Transactions on Mobile Computing*, **19**(9), 2160–2173.

Hekmati, A., Teymoori, P., Todd, T. D., Zhao, D., and Karakostasy, G. (2019b). Optimal multi-decision mobile computation offloading with hard task deadlines. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–8. IEEE.

Hekmati, A., Teymoori, P., Todd, T. D., Zhao, D., and Karakostas, G. (2020). Optimal multi-part mobile computation offloading with hard deadline constraints. *Computer Communications*, **160**, 614–622.

JR, M. N. and Lutimath, N. M. (2020). Eakos-energy aware kmeans offloading scheme for mobile applications. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 280–283. IEEE.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2010). Cuckoo: a computation offloading framework for smartphones. In *International Conference on Mobile Computing, Applications, and Services*, pages 59–79. Springer.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE Infocom*, pages 945–953. IEEE.

Lei, L., Sengupta, S., Pattanaik, T., and Gao, J. (2015). Mclouddb: A mobile cloud database service framework. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 6–15. IEEE.

Li, H., Shou, G., Hu, Y., and Guo, Z. (2016). Mobile edge computing: Progress and challenges. In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pages 83–84. IEEE.

Li, H., Huang, H., and Qian, Z. (2021). Latency-aware batch task offloading for vehicular cloud: Maximizing submodular bandit. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 584–593. IEEE.

Li, Y., Xia, S., Cao, B., Liu, Q., *et al.* (2022). Lyapunov optimization based trade-off policy for mobile cloud offloading in heterogeneous wireless networks. In *IEEE Transactions on Cloud Computing*, number 1, pages 491–505. IEEE.

Liang, F. (x). A survey for the mobile application offloading. *x*.

Liu, D., Khoukhi, L., and Hafid, A. (2017). Data offloading in mobile cloud computing: A markov decision process approach. In *2017 IEEE international conference on communications (ICC)*, pages 1–6. IEEE.

Liu, X., Li, Y., and Chen, H.-H. (2016). Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing. *IEEE Transactions on Vehicular Technology*, **65**(11), 9247–9259.

Mazza, D., Tarchi, D., and Corazza, G. E. (2014). A partial offloading technique for wireless mobile cloud computing in smart cities. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE.

Mukherjee, A., Bhattacherjee, S., Pal, S., and De, D. (2013). Femtocell based green power consumption methods for mobile network. *Computer Networks*, **57**(1), 162–178.

Neely, M. J., Modiano, E., and Rohrs, C. E. (2003). Dynamic power allocation and routing for time varying wireless networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 1, pages 745–755. IEEE.

Niu, J., Song, W., and Atiquzzaman, M. (2014). Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications. *Journal of Network and Computer Applications*, **37**, 334–347.

Orsini, G., Bade, D., and Lamersdorf, W. (2016). Cloudaware: A context-adaptive middleware for mobile edge and cloud computing applications. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 216–221. IEEE.

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

Phillips, R. and Kokotovic, P. (1981). A singular perturbation approach to modeling and control of markov chains. *IEEE Transactions on Automatic Control*, **26**(5), 1087–1094.

Qureshi, S. S., Ahmad, T., Rafique, K., *et al.* (2011). Mobile cloud computing as future for mobile applications-implementation methods and challenging issues. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 467–471. IEEE.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving portable computer battery power through remote process execution. *ACM SIGMOBILE Mobile Computing and Communications Review*, **2**(1), 19–26.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2013). Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, **16**(1), 369–392.

Sardellitti, S., Scutari, G., and Barbarossa, S. (2015). Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, **1**(2), 89–103.

Sharma, R., Kumar, S., and Trivedi, M. C. (2013). Mobile cloud computing: A needed shift from cloud to mobile cloud. In *2013 5th International Conference and Computational Intelligence and Communication Networks*, pages 536–539. IEEE.

SHRM (2021). Google extends work-from-home policy through june 2021. `https://www.shrm.org/resourcesandtools/hr-topics/technology/pages/google-extends-work-from-home-policy-through-june-2021.aspx`.

Statista (2021). Forecast number of mobile devices worldwide from 2020 to 2025 (in billions)*. `https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/`.

Terefe, M. B., Lee, H., Heo, N., Fox, G. C., and Oh, S. (2016). Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*, **27**, 75–89.

Teymoori, P. (2022). Efficient mobile computation offloading with hard task deadlines and concurrent local execution. `http://hdl.handle.net/11375/26482`.

Teymoori, P., Hekmati, A., Todd, T. D., Zhao, D., and Karakostas, G. (2021). Preemptive mobile computation offloading with hard deadlines and concurrent local execution. *IEEE Transactions on Green Communications and Networking*, **5**(2), 950–963.

Uysal-Biyikoglu, E., Prabhakar, B., and El Gamal, A. (2002). Energy-efficient packet transmission over a wireless link. *IEEE/ACM Transactions on Networking*, **10**(4), 487–499.

Wang, C., Li, Y., Jin, D., and Chen, S. (2016). On the serviceability of mobile vehicular cloudlets in a large-scale urban environment. *IEEE Transactions on Intelligent Transportation Systems*, **17**(10), 2960–2970.

Wu, H. and Wolter, K. (2017). Stochastic analysis of delayed mobile offloading in heterogeneous networks. *IEEE Transactions on Mobile Computing*, **17**(2), 461–474.

Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., and Chan, A. (2013). A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, **40**(4), 23–32.

Yang, P., Li, L., Liang, W., Zhang, H., and Ding, Z. (2019a). Latency optimization for multi-user noma-mec offloading using reinforcement learning. In *2019 28th Wireless and Optical Communications Conference (WOCC)*, pages 1–5. IEEE.

Yang, P., Li, L., Liang, W., Zhang, H., and Ding, Z. (2019b). Latency optimization for multi-user noma-mec offloading using reinforcement learning. In *2019 28th Wireless and Optical Communications Conference (WOCC)*, pages 1–5. IEEE.

Yang, Y., Geng, Y., Qiu, L., Hu, W., and Cao, G. (2017). Context-aware task offloading for wearable devices. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE.

Yi, C., Cai, J., and Su, Z. (2019). A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. *IEEE Transactions on Mobile Computing*, **19**(1), 29–43.

Yin, G. G. and Zhang, Q. (2012). *Continuous-time Markov chains and applications: a singular perturbation approach*, volume 37. Springer.

Zafer, M. and Modiano, E. (2007). Minimum energy transmission over a wireless fading channel with packet deadlines. pages 1148–1155.

Zhan, K., Lung, C.-H., and Srivastava, P. (2014). A comparative evaluation of computation offloading for mobile applications. In *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 518–525. IEEE.

Zhang, K., Mao, Y., Leng, S., Maharjan, S., and Zhang, Y. (2017a). Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

Zhang, K., Mao, Y., Leng, S., Maharjan, S., and Zhang, Y. (2017b). Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O. (2013). Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications*, **12**(9), 4569–4581.

Zhang, W., Wen, Y., and Wu, D. O. (2014). Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Transactions on Wireless Communications*, **14**(1), 81–93.

Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S. (2012). Refactoring android java code for on-demand computation offloading. *ACM Sigplan Notices*, **47**(10), 233–248.