# Algorithms for Tasks Offloading on Multiple Mobile Edge Servers

Houssemeddine Mazouzi

UNIVERSITÉ PARIS 13

Galilée Ecole doctorale

# THÈSE

**pour obtenir le grade de**

## Docteur de l'université Paris 13

**Discipline : "Ingénierie Informatique"**

*présentée et soutenue publiquement par*

**Houssemeddine MAZOUZI**

le 22 Novembre 2019

# Algorithmes pour le déchargement de tâches sur serveurs de périphérie

Directeur de thèse : **Dr. Nadjib Achir**

Co-encadrant de thèse : **Dr. Khaled Boussetta**

### JURY

| | | |
|---|---|---|
| Hervé Rivano, | Professeur, INSA Lyon | Président du Jury |
| Marcelo Dias de Amorim, | Directeur de Recherche CNRS | Rapporteur |
| Stefano Secci, | Professeur, Laboratoire CEDRIC, CNAM Paris | Rapporteur |
| Aline Carneiro Viana, | Chargée de Recherche HDR, INRIA | Examinatrice |
| Jeremie Leguay, | Docteur, Huawei France | Examinateur |
| Rami Langar, | Professeur, Université Paris-Est Marne-la-Vallée | Examinateur |
| Nadjib Achir, | Maître de conférences, Université Paris 13 | Directeur |
| Khaled Boussetta, | Maître de conférences, Université Paris 13 | Co-encadrant |

L2TI LABORATOIRE DE TRAITEMENT ET TRANSPORT DE L'INFORMATION UNIVERSITE PARIS 13

# Thesis

**Submitted for the degree of Doctor of Philosophy of**

## Université Paris 13

**Specialization : "Computer Engineering"**

*presented and defended by*

**Houssemeddine MAZOUZI**

$22^{nd}$ November 2019

# Algorithms for Tasks Offloading on Multiple Mobile Edge Servers

Supervisor : **Dr. Nadjib Achir**

co-supervisor : **Dr. Khaled Boussetta**

### Committee

| | | |
|---|---|---|
| Hervé Rivano, | University Professor, INSA Lyon | Chairman |
| Marcelo Dias de Amorim, | Research Director CNRS | Reviewer |
| Stefano Secci, | Professor, Laboratory CEDRIC, CNAM Paris | Reviewer |
| Aline Carneiro Viana, | INRIA Research Scientist HDR | Examiner |
| Jeremie Leguay, | Ph.D, Huawei France | Examiner |
| Rami Langar, | Full Professor, Université Paris-Est Marne-la-Vallée | Examiner |
| Nadjib Achir, | Professor, Université Paris 13 | Supervisor |
| Khaled Boussetta, | Professor, Université Paris 13 | Co-Supervisor |

*L2TI – EA3043*
*École doctorale Galilée – Université Paris 13*
*99, avenue Jean-Baptiste Clément*
*93430 Villetaneuse.*

# Résumé

Le déchargement de calculs est l'une des solutions les plus prometteuses pour surmonter le manque de ressources au niveau des terminaux mobiles. Elle permet l'exécution d'une partie ou de la totalité d'une application mobile dans le cloud. L'objectif est d'améliorer les temps d'exécution et de réduire la consommation énergétique. Malheureusement, même si les Clouds disposent d'importantes ressources de calcul et de stockage, ils sont généralement éloignés des équipements terminaux. Dans ce cas, cette approche peut souffrir de délais importants et fluctuants. Cette constatation est particulièrement problématique pour certaines applications pour lesquelles un temps de réponse réduit est nécessaire. Pour réduire ce délai d'accès, l'une des approches émergentes est de pousser le Cloud vers la bordure du réseau. Cette proximité permet aux applications mobiles de décharger leurs tâches et données vers un Cloud "**local**" ou "Edge Cloud". Un **Edge Cloud** peut être vu comme un petit centre de traitement des données relativement à de plus grands data centres. Cette proximité géographique entre les applications mobiles et le cloud local signifie que le délai d'accès peut être considérablement réduit par rapport aux Clouds distants, mais également la possibilité d'obtenir un débit plus important, une plus grande réactivité, un meilleur passage à l'échelle, etc.

Dans cette thèse, nous nous concentrons sur le déchargement de calculs dans une architecture de type mobiles (**Mobile Edge Computing** – MEC), composée de plusieurs serveurs de périphérie. Notre objectif est d'explorer de nouvelles stratégies de déchargement efficaces afin d'améliorer les performances des applications tant du point de vue délais de calcul que consommation énergétique, tout en garantissant les contraintes de temps d'exécution des applications. Notre première contribution est une nouvelle stratégie de déchargement sur plusieurs serveurs de périphérie. Nous proposons par la suite une extension de cette stratégie en incluant également le Cloud. Nous évaluons ces stratégies tant du point de vue théorique que pratique avec l'implémentation d'un middleware de déchargement. Finalement, nous proposons une nouvelle approche élas-

tique dans le cas d'applications multitâches caractérisées par un graphe de dépendances entre les tâches.

# Abstract

Computation offloading is one of the most promising paradigm to overcome the lack of computational resources in mobile devices. Basically, it allows the execution of part or all of a mobile application in the cloud. The main objective is to reduce both execution time and energy consumption for the mobile terminals. Unfortunately, even if clouds have rich computing and storage resources, they are usually geographically far from mobile applications and may suffer from large delays, which is particularly problematic for mobile applications with small response time requirements. To reduce this long delay, one of the emerging approach is to push the cloud to the network edge. This proximity gives the opportunity to mobile users to offload their tasks to "**local**" cloud for processing. An **Edge Cloud** can be seen as small data center acting as a shadow image of larger data centers. This geographical proximity between mobile applications and edge cloud means that the access delay can be greatly reduced, but affects also higher throughput, improved responsiveness and better scalability.

In this thesis, we focus on computation offloading in mobile environment (**Mobile Edge Computing** - MEC), composed of several edge servers. Our goal is to explore new and effective offloading strategies to improve applications performances in both execution time and energy consumption, while ensuring application requirements. Our first contribution is a new offloading strategy in the case of multiple edge servers. Then we extend this strategy to include the Cloud. Both strategies have been evaluated theoretically and experimentally by the implementation of an offloading middleware. Finally, we propose a new elastic approach in the case of multitasking applications characterized by a graph of dependencies between tasks.

# Acknowledgments

This PhD project was a great opportunity to improve myself. I would like to express my sincere thanks to :

*Nadjib ACHIR and Khaled BOUSSETTA*, my supervisors, who always have ideas and think out of the box, for their support, enthusiasm, good comments to my thesis, and constant technical guidance.

*Marcelo Dias de Amorim and Stefano Secci*, my manuscript reviewers, for their effort and time to review my dissertation, for their appreciation of my work. Also for their positive remarks that improve the quality of my manuscript.

*Aline Carneiro Viana*, *Hervé Rivano*, *Jeremie Leguay*, and *Rami Langar*, my examiners, for accepting to evaluate the work of my PhD, their time and effort to prepare my thesis defense.

All my friends in L2TI laboratory, of université Paris 13, for their encouragement and support. Thanks to them, my stuy and time in this laboratory was more enjoyable. I am happy to share a lot of memorable moments with them.

All family members for their endless support and motivation. In addition to all nice people who I had many stories during the course of this three years.

Thank you very much, everyone !

# Contents

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| AP | Access Point |
| AWGN | Additive White Gaussian Noise |
| BL-EST | Bottom Level Early Start Time |
| CBL | edge server-Cloudlet- load Balancing seLection heuristic |
| CPU | Central Pprocessing Unit |
| DAG | Directed Acyclic Graph |
| DM2-ECOP | Dynamic Multi-user Multi-edge server Efficient Computation Offloading Policy |
| DOTA | Delay bounded Optimal egde server-cloudleT- user Association heuristic |
| ECESO | Efficient edge server-Cloudlet- Selection Offloading policy |
| eRAM | elastic Remote task Acceleration for resource-constrained Mobile terminals |
| eRAM-OC | eRAM Offloading Client |
| eRAM-OS | eRAM Offloading Server |
| eTOMEC | elastic Task graph Offloading in Mobile Edge Computing |
| FCO | Full Cloud Offloading policy |
| GAP | General Assignement Problem |
| GBC-SFH | Greedy Best Cloudlet Selection First Heuristic |
| HEM | Heavy Edge Merge |

| | |
|---|---|
| **LAH** | **L**agrangian **A**djustment **H**euristic |
| **MCC** | **M**obile **C**loud **C**omputing |
| **MCOP** | **M**inimun **C**ost **O**ffloading **P**artitioning |
| **MEC** | **M**obile **E**dge **C**omputing |
| **NCO** | **N**earset edge server-**C**loudlet- **O**ffloading policy |
| **OFDMA** | **O**rthogonal **F**requency-**D**ivision **M**ultiple **A**ccess |
| **QAM** | **Q**uadrature **A**mplitude **M**odulation |
| **QoS** | **Q**uality **o**f **S**ervice |
| **RTT** | **R**ound **T**rip **T**ime |
| **SNR** | **S**ignal-to-**N**oise **R**atio |
| **WFM** | **W**ave **F**ront **M**ethod |

# Notations

| | |
|---|---|
| K | the number of edge servers available in the network. |
| M | the number of APs in the network. |
| $N_m$ | the number of users associated to the AP m. |
| $f_{m,n}$ | the local computing capacity of the $n^{th}$ user of the $m^{th}$ AP. |
| $F^k$ | the computing capacity of the edge server k. |
| $c^k$ | the computing resource allocation on edge server k. |
| $(m,n)$ | the $n^{th}$ user of the $m^{th}$ AP in the MEC. |
| $dw_{m,n}$ | the amount of data to download by the user $(m,n)$. |
| $up_{m,n}$ | the amount of data uploaded to the MEC from the user $(m,n)$. |
| $w^{tx}_{m,n}$ | the allocated upload data rate for the user $(m,n)$. |
| $w^{rx}_{m,n}$ | the allocated download data rate for the user $(m,n)$. |
| $p^{tx}_{m,n}$ | power consumption when the network interface is transforming data. |
| $p^{rx}_{m,n}$ | power consumption when the network interface is receiving data. |
| $p^{idle}_{m,n}$ | power consumption when the network interface is in Idle state |
| $t_{m,n}$ | the maximum tolerated delay according the QoS of the task of the user $(m,n)$. |
| $T^t_{m,n,k}$ | the communication time when user $(m,n)$ offload to edge server k. |
| $T^l_{m,n}$ | the local processing time for user $(m,n)$. |
| $T^{e,k}_{m,n}$ | the remote processing time for user $(m,n)$ in edge server k. |
| $\mathcal{Z}^l_{m,n}$ | the local energy consumption for user $(m,n)$. |
| $\mathcal{Z}^{e,k}_{m,n}$ | the remote energy consumption for user $(m,n)$ in edge server k. |
| $\gamma_{m,n}$ | the computational resource required by the task of user $(m,n)$. |
| $\lambda_m$ | the Lagrangian multiplier of the subproblem m. |

| | |
|---|---|
| $x^k_{m,n}$ | the offloading decision variable for the task of user $(m,n)$ in the edge server k. |
| $y_{m,n}$ | the category to which belong the tasks (static or dynamic offloading decision task). |
| $g^k_{m,n}$ | variable if the edge server $k$ can perform the task of user $(m,n)$. |
| $a_{m,n}$ | the amount of computation that user $(m,n)$ should offload to MEC. |
| | |
| $s_i$ | the code source of the task $i$ in multitasking application. |
| $e_{j,i}$ | the data dependency between tasks $i$ and $j$ in multitasking application. |
| $st^k_i$ | the start time of the task $i$, in multitasking application, on the edge server $k$. |
| $ft_{j,i}$ | the finish time of the task $i$, in multitasking application, on the edge server $k$. |
| $bl(i)$ | the priority of task $i$ in multitasking application. |

Chapter 1

# Introduction

Recently, mobile terminals have undergone a great transformation from small devices with limited resource capabilities to essential everyday accessories. They are indispensable for our daily communications, personal and professional activities. The number of those terminals has increased dramatically, with a growth in their applications and uses. However, the willingness to limit the weight and the size by the constructors restricts seriously the resource capability and the battery lifetime. Consequently, a powerful approach to enhance the performance of applications and alleviate the resource scarcity of mobile terminals is to enable the mobile terminal to perform some or all of their computational-intensive application to a remote high-performance server, such as the cloud, and known as computation offloading. In this chapter, we present the motivation and challenges of the computation offloading to a new emerging computing paradigm known as Mobile Edge Computing – MEC.

## 1.1 Context

The major progress in the mobile terminals technologies has facilitated the emergence of innovative mobile applications, which integrate richer functionalities and features such as augmented/virtual reality, face/speech recognition or high-performance operations. To fulfill the computing resource demands of these applications, mobile terminals constructors, such as Apple, Huawei, and Samsung, increase the build-in computing capabilities

of these terminals. Nevertheless, those novel applications are often computationally intensive and resource-hungry. Consequently, the gap between the required and the available resources continues to grow. As a result, the users quality of experience is highly affected and can suffer from poor latency.

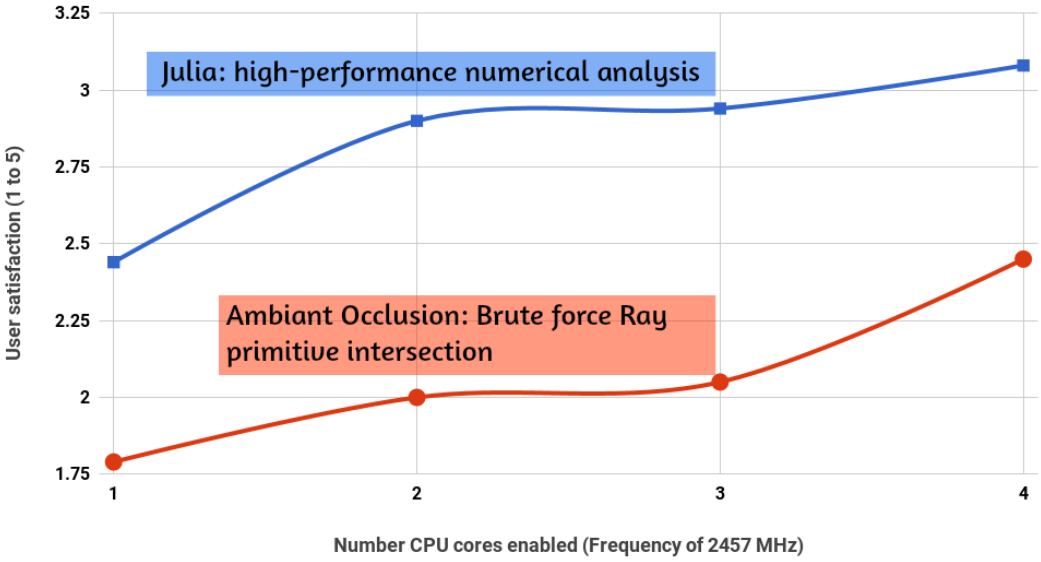



**Figure 1.1:** *User satisfaction for the performance of the applications on Samsung Galaxy S5 smartphone. (1) Very unsatisfied (5) Very satisfied [1].*

Figure 1.1 draws user satisfaction of the performance of two resource-hungry applications I) *Julia a high-performance numerical analysis*, and II) *Ambiant Occlusion a brute force ray primitive intersection* as presented in [1]. The experiments are done using a Samsung Galaxy S5 smartphone with a computing capacity of 4 CPU-cores, each has a frequency of 2.5 GHz. The User then gives his satisfaction in a scale from `1- very unsatisfied` to `5- very satisfied`. We note that users become more satisfied when the computing resource increases. Nevertheless, user's satisfaction is below than `3- Neither Satisfied Nor Dissatisfied` even when the application uses all the computing resources of the smartphone. Consequently, the quality of experience of such applications can hardly fulfilled. So, it is very important to circumvent this limitation of the mobile terminals capabilities to meet the needs of resource-hungry applications.

One possible approach is to perform these heavy computations by a more powerful system outside the mobile terminal, which is referred to `computation offloading`. The latter consists of delegating partially or completely the computation to a resource-powerful remote server such as cloud computing. In the past years, many works have

proposed different approaches and solutions of computation offloading including load sharing [2], remote execution [3], and cyber foraging [4] to improve the performance of the application and user experience of mobile terminals. The success of cloud computing as a powerful environment with unlimited computing capabilities and the evolution of wireless technologies that makes network connectivity ubiquitous motivate several works to bring the cloud computing capacities to the mobile terminals from offloading perspective. Consequently, `Mobile Cloud Computing (MCC)` [5] was introduced to enrich the mobile terminals computation and battery lifetime issues. The benefits of MCC can be summarized as the following [4]:

- **`Enhanced computing capability:`** Offloading some or all intensive computations of the application to a reliable computing resource increases significantly the processing capability of the mobile terminal, which reduces the execution time.
- **`Increased battery lifetime:`** Moving the computation to cloud can decrease energy consumption and increases mobile terminal battery lifetime.
- **`Unlimited data storage:`** Using the unlimited storage capacity of the cloud, to save applications' data, can save the storage capability of the mobile terminal.
- **`Enriched user interface:`** Due to graphical resources in the cloud, such as GPU, MCC heavily impact 2D and 3D rendering of the application. Consequently, The users experience richer features for multimedia applications.

Although MCC improves the performances of the application, the geographical distance to the remote cloud might induce significant delays that are not tolerated by many time-sensitive applications, such as online mobile gaming, augmented reality, face and speech recognition [6, 7]. Consequently, the user may suffer from poor experience with those applications. To address this issue, the Mobile Edge Computing (MEC) paradigm has recently emerged as a credible solution to support resource-hungry and time-sensitive applications. The fundamental concept underlying MEC is to push the computing capabilities at the edge of the network, closer to end-users rather than in the cloud. Concretely, a typical MEC environment is composed of small data-centers, called edge servers, that are deployed nearby wireless APs, ADSL boxes or Base Stations (BSs). Each edge server offers higher processing capacities than most mobile terminals, allowing thus a faster execution of offloaded tasks. This geographical proximity between

mobile terminals and edge servers means that the MEC access delay on task offloading can be greatly reduced, compared to MCC, thereby significantly improving mobile user experiences. Moreover, the mobile terminal can offload its computation and data with ultra-high bandwidth compared to MCC. However, the MEC is composed of a small server which means that each edge server can perform a limited computation at a time. As a result, MEC performances will be decreased in a multi-user scenario when many users try to offload their computations to the MEC. Consequently, the existing computation offloading policies must be enhanced to improve the benefits of the MEC in such scenarios.

## 1.2 Challenges

Enhancing the performances of computation offloading on MEC environment remains an important issue. In recent years, many works presented different models and architectures to offload computation-intensive applications to MEC. Most of these works conclude that the efficiency of the offloading to MEC relies on many dimensions [4]. Identifying these dimensions results in designing an offloading framework that efficiently improves the performance of mobile terminals and the applications [8].

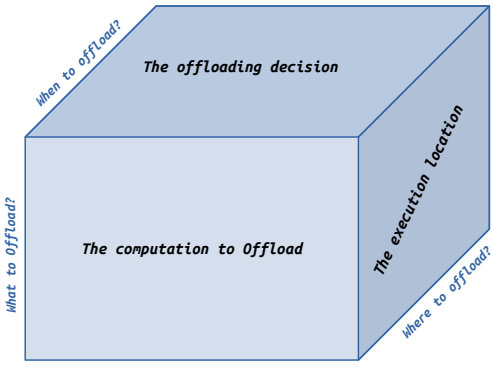



**Figure 1.2:** *Computation offloading dimensions [4].*

As detailed in Figure 1.2, several works define three main dimensions of computation offloading [4, 8]. Each dimension refers to a specific challenge of the offloading. Addressing these dimensions determines how to offload the computation from a mobile terminal to a remote server (MCC/MEC) and can be summarized as follows:

- The first dimension address the computation to offload, it gives answer to the question of what to offload? Existing works proposed three main approaches:

1) **The application:** the hole application is offloaded, then the results will be sent back to the mobile terminal.
2) **A task:** in this case, the application is partitioned into smaller entities, called tasks, then the mobile offload some or all the tasks depending on the network and computing resources availability.
3) **The mobile system image:** an overlay VM image of the mobile operating system, for example Android, and the application will be transmitted to an edge server in MEC.

- The second dimension targets the offloading decision, it tries to answer the question when to offload? The offloading decision is the decision whether the chosen computation will be offloaded. We define two categories for the offloading decision as following:

  1) **Static offloading decision:** here the offloading decision is done at the design time of the application. The offloadable part of computation will always be performed remotely what ever the conditions and the situation of the mobile terminal.
  2) **Dynamic offloading decision:** here the offloading decision is token at runtime of the application depending on the network and system resource availability.

- The last dimension investigates the question of where to offload? Choosing the execution place of the computation is a challenging issue. It depends on to many factors including the application QoS requirement and the computing resource availability. In a MEC environment three possible locations are available:

  1) **The mobile terminal:** in this case there is no offloading and the computation is performed locally within the mobile terminal.
  2) **The edge server(s):** in the case where the mobile terminal decide to offload its computation to edge server(s). It is clear that the location of the target edge server(s) has a great impact on the performances experienced by the end user.
  3) **The remote cloud:** finally, the mobile terminal can also offload its computation directly to the remote cloud.

Most of existing studies focused on computation offloading of mobile terminals to one target place: either to a selected edge server or to the remote cloud, assuming that the execution place has already been chosen statically [4, 8]. However, in real-world scenario this preselected execution place maybe overloaded and cannot perform new tasks. Moreover, several edge servers might be available around the mobile terminal. The selection of to which server the terminal should offload its tasks is a challenging issue, and have a big impact on the performance of the application and the overall MEC environment.

In this thesis, we focus on the computation offloading in multi-edge server MEC environment, where an Internet Service Provider has deployed several servers acting as edge cloud close to existing access points or base stations. More precisely, we focus on the selection of best offloading server(s) where we should perform the computation of the mobile terminals. It is clear, that poor offloading server selection will result in heavily unbalanced load among the servers. On the other hand, an efficient selection of the offloading servers can significantly improve the performances of the applications and thus increases the quality of experience for the end-users. The resulting challenges are:

1) which mobile terminals should offload their computation?

2) which parts of the application should be offloaded?

3) which server (edge/cloud) should perform the computation and of which mobile terminal?

In addition, the last challenges we were also interested in the impact of application pattern on the offloading efficiency.

## 1.3 Contributions

We start by investigating the problem of the minimization the energy consumed by the mobile terminal. Basically, we design an energy-efficient offloading policy that decides which users should offload and selects the best edge server where to perform that offloading computations? In addition, we also investigated the impact of the amount of offloaded computation on the offloading performances. Finally, because we consider that all the applications have the same priority and also because they are not sharing resources at the offloading server, we were able to demonstrate that the best choice is to offload all the computation to the MEC or perform all of it locally.

Second, we extended the last proposal by including the remote cloud as a candidate offloading server, and we propose a two-tier offloading policy. In addition to the theoretical analysis, we also added a complete experimental approach in order to evaluate our policy with real Android mobile applications. In this case, we designed and implemented a computation offloading middleware for Android-based terminals. This middleware was integrated on both the mobile terminal and the offloading server. Using this testbed we were able to run multi-users offloading experiments in local edge and in the cloud, with the aim to compare the observed performances to the placement decision made by our proposal.

Finally, the last contribution focuses on the offloading of multitasking applications, by investigating the impact of the application's architecture on the offloading performances. We address this issue through a mobile user's perspective that is seeking to obtain the execution result of a resource-hungry multitasking application, possibly through offloading some or all the tasks to a multiple edge servers'. Basically, we decide which are the tasks that should be offloaded according to the task-dependency graph of the application.

## 1.4 Thesis outline

In this chapter, we present the computation offloading new challenges and issues for MEC environment. The rest of this manuscript is organized as follows:

- **Chapter 2 Backgrounds:** This chapter provides the backgrounds required in order to understand our contributions in this thesis.
- **Chapter 3:** This chapter investigates computation offloading in multi-user multi-edge server MEC environment. A new offloading policy was introduced in order to improve the performance of the offloading in such environment.
- **Chapter 4;** This chapter discusses an extended version of our first offloading policy to consider a two-tired MEC environment. In addition, it explores the real-world experiments to prove the validity of our proposal.
- **Chapter 5:** This chapter explores the computation offloading for multitasking application in multi-edge server MEC environment and it presents an efficient offloading policy that consider offloading of graph-based dependent tasks to multiple edge servers.

- **Conclusion 6:** This chapter draws a conclusion of this thesis and presents some perspectives and future works.
- **Appendix A:** This appendix provides the proof of the theorems proposed in this thesis.

CHAPTER 2

# Backgrounds

THIS chapter illustrates the main motivations and issues behind the importance of the MEC paradigm. It then discusses the principal features of this emerging computing paradigm. Finally, the chapter investigates the major limitations and constraints essential to the deployment of MEC in an existing network infrastructure. In addition, the chapter presents the state-of-the-art and the challenges of computation offloading in MEC. These considerations are viewed as prerequisites for an understanding of the remaining work presented in this thesis.

Chapter 2 is structured as the following. Section 2.1 discusses the characteristics of the MEC architecture and the benefits of this new environment for applications. In section 2.2, we present computation offloading taxonomy of the works in the literature. Existing works and problems of computation offloading are presented in section 2.3. The last section 2.4 draws a conclusion about the importance of MEC environment for computation offloading and the main challenges and issues to address in order to enhance the performances of the existing solutions.

## 2.1 Overview of MEC architecture

MEC has emerged as a main paradigm to extends the cloud resource capabilities to the edge of the network [9, 10]. Several proposals were introduced in order to implement the MEC in a real-world scenario. Despite the differences in the literature definitions, MEC has been commonly regarded as a set of small servers located at the APs or Base

stations, which justify its partitioning as a two-tier cloud [4]. Initially, MEC has been introduced to reduce the network delay between the mobile terminals and the remote cloud. It is mainly used in the purpose of computation and data offloading, where mobile applications and services are performed with low latency near to the end-user. Consequently, a huge number of computation offloading approaches and policies has been proposed to improve the mobile terminal performance in terms of completion time and energy consumption of the applications. Moreover, recent works enrich the MEC by a lots of features and functionalities such as: mobile web browser acceleration [11], Healthcare [12], connected vehicles [13, 14] and video streaming & analysis [15].

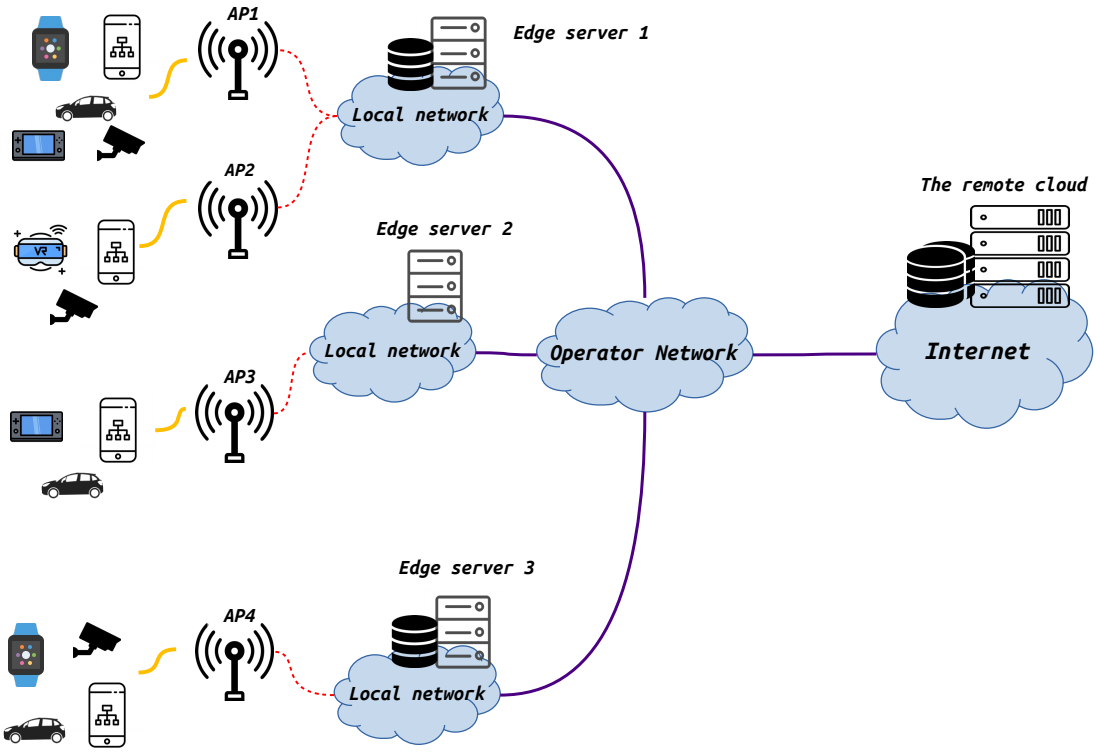



**Figure 2.1:** *Overview of the architecture of a real-world scenario MEC environment.*

Various architectures of this paradigm has been introduced to study its effect on real-world deployment [4]. We distinguish three implementations each one defines how to deploy a MEC servers within existing network infrastructure.

**`The Fog nodes`** [16] are heterogeneous edge servers that can be placed anywhere between the cloud and the end users. The heterogeneity of fog nodes makes this architecture capable of supporting a variety of devices working on different network and virtualization implementations.

Alternatively, a **`cloudlet`** [8] is a VM based edge server located at the edge of

an existing network. Cloudlet environment can be two-tiers and benefits from the cloud capacity. While the cloudlets address the challenges of bandwidth demand and in-advance streaming using high Internet bandwidth, the fog node has more features for IoT and small devices. However, the owner edge server in both of cloudlet and fog node can be anyone.

The third implementation is the standardization designed by **ESTI group** [17]. Unlike the first two implementations, **ESTI group** defines algorithms and standards that describe how to develop, deploy and use the MEC infrastructure. It supposes that edge servers will be deployed in existing APs or base stations, and that the network's operator owned all the edge servers.

In this thesis, we use the **ESTI group** standardization to consider a real-world scenario MEC environment, which is a two-tiered cloud as illustrated in Figure 2.1. First, a set of edge servers deployed at the APs and owned by the network's operator that serve end users devices such as connected cars, smartphones, smartwatches, etc. Each edge server can connect to the remote cloud through the Internet in order to benefit from the unlimited cloud resources. **ESTI standardization** of MEC has the following advantages:

- **On-demand self services:** It ensures that MEC offers resources to mobile terminals in few seconds. For some complex applications and services, end-users could be invited to proceed an anticipated computing resource reservation via a web-portal.

- **Large variety of access technologies:** It enables every end-user to access computing services from various types of devices, either fixed, e.g: personal computers and workstations, or mobile terminals like smartphones and tablets or IoT devices.

- **Elasticity:** Extended from the MCC paradigm, MEC resources are provisioned and released on-demand to the end-user in an elastic way. Elasticity of resources consists in enabling the end-user to increase or to decrease the allocated resources used by their tasks.

- **Energy efficiency:** The hardware component of mobile terminal such as screen, processor and network interfaces are more energy demanding, one of the most

important issue of nowadays mobile terminals is the battery lifetime. Since manufacturers cannot meet the growing demands, MEC can be a promising alternative solution by bringing the cloud computing resources to the mobile terminals' proximity to enhance its battery lifetime.

- **`Proximity:`** By deploying resources near to end-users, MEC has an advantage for latency-sensitive applications and compute-hungry devices, such as augmented reality (AR), video analytics, etc.
- **`Lower Latency:`** Edge servers are deployed at the nearest location to end users, isolating network data movement from the core network. Hence, user experience is accounted high quality with ultra-low latency and high bandwidth.
- **`Localization awareness:`** Edge-distributed devices utilize low-level signaling for information sharing. MEC receives information from edge devices within the local access network to discover the location of devices.
- **`Soft state:`** In the standardization every edge server does not have any hard state, which means that the server contains only cached states of the VMs of the applications and servers. It must download the images of this VMs form the remote cloud or the mobile terminal. In addition, users' data can be buffered in edge servers en route to safety in the cloud.

Technically and economically speaking MEC is a very promising extension of the cloud computing. It can enhance considerably the performances of a huge bunch of applications and services, and reinforce the quality of the user's experience. One of the main advantages to this paradigm is to move several applications from prototyping to development and large-scale usage such as: mobile gaming and mobile VR/VA applications. MEC provides high distribution of the computing resources, so end-user can run where it senses better to him. However, a real-world deployment and usage of MEC requires to solve challenging issues related to the performances, the applications architecture and the execution location.

## 2.2 Taxonomy of computation offloading

The emergence of MEC allows computation offloading of applications from poor-resource mobile terminals. Many works have investigated this issue, introducing several new

approaches with different offloading policies and algorithms. In this section, we will present computation offloading taxonomy of the state-of-the-art on existing approaches. The taxonomy refers to the techniques, methods and assumptions considered in the literature.

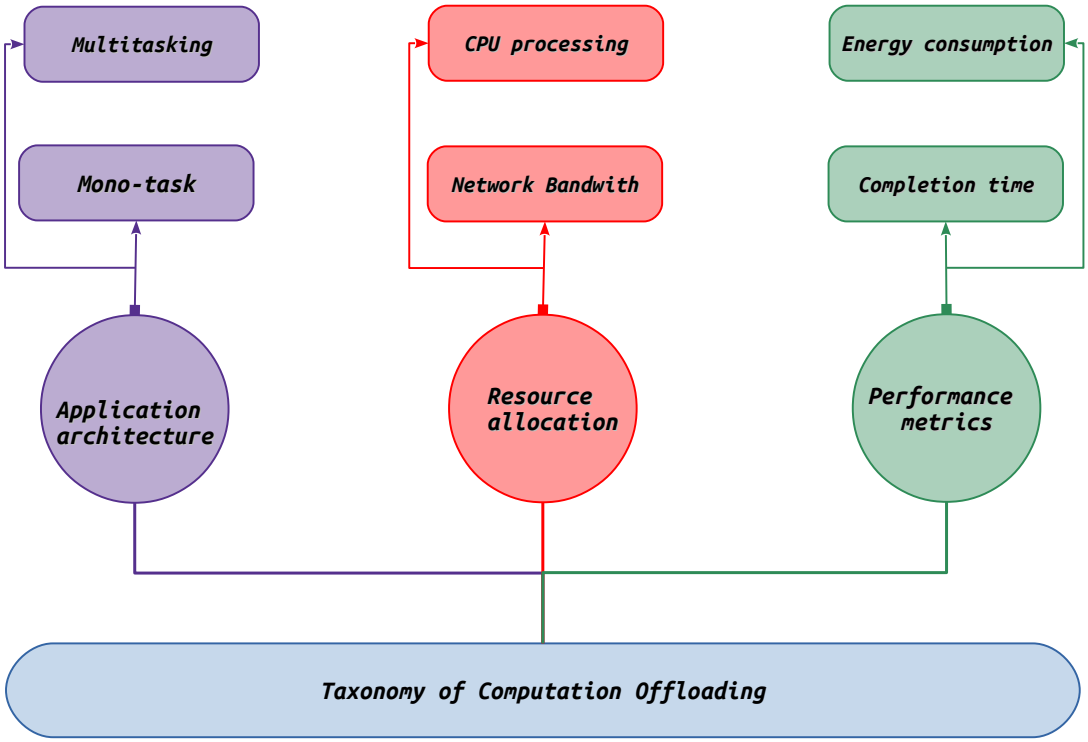



**Figure 2.2:** *Taxonomy of computation offloading*

As shown in Figure 2.2, the taxonomy of computation offloading has the following three aspects:

- **`The application architecture:`** This aspect defines the architecture of the application that is supported by the MEC. This architecture determines how the application will be offloaded, and describes which parts could be offloaded. A task is a part that can be offloaded to a remote server such as: MCC or MEC. It can be a module, a class, a thread or even a method of the application. We recognize two categories of applications supported by a MEC environment: i) `the mono-task application`, which has only on task that can be offloaded. And, ii)`the multitasking application` that is composed of many tasks, which can be executed locally or remotely in the MEC. Moreover, the tasks of an application can be independents, or related to each other through a task-dependency graph.

- **`The resource allocation:`** A MEC application requires two types of resource: i) for its data volume, and ii) for its computation processing. The data volume refers to the source code of the application, its inputs and outputs data. An offloaded application needs network resource to transmit the data volume to the target server. The network resource corresponds to the network bandwidth, so the mobile terminal must reserve an amount of the bandwidth to transmit its data to the MEC servers. In addition, the application requires CPU processing resources to process its computation in the target server in MEC. Consequently, the mobile terminal must allocate CPU resources in the target edge server to perform its offloaded applications.

- **`The performance metrics:`** This aspect is related to the performances measurement of the offloading. The metric is the main indicator of the offloading decision. Literature works introduced two metrics: i) `the completion time` of the application that measures the time from when an application start till it finishes include the transmission time, the processing time and the preparation time to display or store the output of the application. This metric can also indicate the quality of experience of the user with the application, obtaining a shorter completion time means that the user will be very satisfied with the performance of this application. Consequently, minimizing this metric is important for the efficiency of the computation offloading. An other metric is : ii) `the energy consumption`, which measures the amount of the energy consummated by the mobile terminal during the performing of the application. It can indicate the duration of the autonomy of the mobile terminal, and so defines the duration of usage of the device. A lower energy consumption makes the battery lifetime longer, the device will be autonomous for long time and the user can use it longer. As consequence, reducing this metric has a great impact on the usability of the mobile terminal. These two metrics can be used each alone or as a linear combination.

## 2.3 State-of-the-art of computation offloading

An offloading work can address one or more dimensions of the computation offloading defining previously in the Figure 1.2. Depending on the objective of the computation offloading in MEC, an offloading policy investigates one or more challenging issues and problems. In Figure 2.3, we distinguish four main issues addressed by the existing

works. As we can notice, the approaches in Edge server placement and Edge server Selection categories investigate what to offload and where to offload dimensions. However, categories of Offloading decision and Application Partitioning study the when to offload and What to offload and where to offload dimensions.

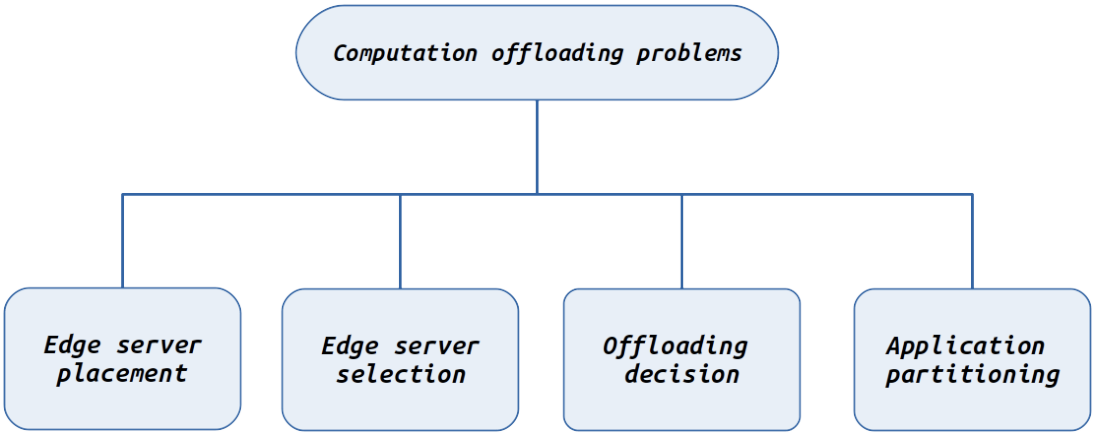



**Figure 2.3:** *Review of computation offloading problems.*

### 2.3.1 Edge server placement problem

Edge servers placement problem refers to the deployment of the servers in a real environment. It addresses the issues related to the deployment cost, the effect on exiting network infrastructure, the efficiency of the deployment for the end-users, and the maintenance cost of the environment. Mike Jia et al. [18, 19] offloading approach is one of the first heuristics on edge servers placement in a large-scale MEC environment. Its main goal is to find the best edge servers placement in a large network, then select an edge server to perform the computation associated to each AP. The heuristic uses K-median clustering based on user density, then each AP is statically assigned to one edge server. Similarly, Hong Yao et al. [20] introduced edge servers placement and selection algorithms to support heterogeneous multi-user multi-edge server MEC environment. Longjie Ma et al. [21] introduced a heuristic to find the minimal number of edge servers that must be placed in an existing network infrastructure in order to improve the user experience by performing more offloaded tasks. The users associated to the same AP are assigned to the same edge server. In similar way, Hyun-sun Lee et al. [22] designed an edge servers deployment heuristic to maximize the network's operator profile while satisfying the user's Quality of Service (QoS). The objective was to generate more revenue for the network's operator by deploying the optimal number

of edge servers within the existing AP. Finally, Yuanzhe Li et al. [23] implemented a heuristic to deploy a set of edge servers within a local regions of APs. The objective is to find the optimal placement solution that minimize the access time to the edge server, and assign each AP to one edge server. Even if these proposals define a near optimal way to deploy the servers of MEC, there is many problems of the offloading still not investigated such as: determine the tasks to offload and choose another server when to preassigned one is overloaded. Consequently, an efficient offloading policy must address this problems.

### 2.3.2 Edge server selection problem

Other works try to find a dynamic edge server selection in multi-edge server MEC environment, where multiple edge servers are available around mobile terminals, in order to improve the performance of large scale MEC. Anwesha Mukherjee et al. [24, 25] designed a multi-level offloading policy to optimize the energy consumption. The users offload to the nearest edge server in the first step. According to the amount of resource available in this edge server, it can perform the tasks or offload them that to another edge server and so on. Mike Jia et al. [26] introduced a heuristic to balance the load between the edge server. Its main goal is to migrate some computations from overloaded edge servers to underloaded edge servers to reduce the execution time and hence improve the performance of the computation offloading in the MEC. In the same way, Qiliang Zhu et al. [27] developed a two-tier offloading policy, where the mobile terminal offloads its computation to a server based on the resources availability. The policy used an agent that decides to perform the computation in the local edge server or to offload it to the cloud. Arash Bozorgchenani et al. [28] offloading policy tries to select a nearby fog node to offload some computation of a busy fog node in order to save energy consumption and completion time. Similarly, Xile Shen et al. [29] introduced an offloading policy to select the best edge server to offload to in order to reduce completion time of task. It predicts the network and system status to select to best edge server. Finally, Tao Ouyang et al. [30] studied services placement in MEC environment in order to minimize the completion time of the tasks. The objective is to select the edge server to perform the tasks, service, of each user using machine learning approach. Even these works proposed dynamic edge server selection heuristics, the tasks is always offloaded to the nearest edge server that decides to perform them locally or transmit them the

other edge servers. Thus, an additional offloading cost is induced; consequently, the performance of offloading will decrease. To improve the performances of these works, the edge server that performs each offloaded task will be determined at the offloading decision time without any additional costs.

### 2.3.3 Offloading decision problem

As we discuss previously, the offloading decision has a crucial role to determine the efficiency of the offloading policy. Many recent works investigate the offloading decision on MCC or MEC environment, they consider a multi-user scenarios where several mobile terminals are connected to the AP, each has a resource-hungry task and needs to offload it to the MCC/MEC. The objective is then is to decide which tasks should be offloaded and which one are performed locally.

The first category of works focuses on multi-user single remote server scenario. Meng-Hsi et al. [31] is one the first work on multi-user computation offloading in MCC. The proposed offloading policy determines which computation must be performed in the cloud and which one must be performed locally, by the mobile terminal. Then, it allocates the wireless bandwidth to each user to reduce the energy consumption of the mobile terminal. Xu Chen et al. [32] offloading policy was designed for a single edge server MEC environment. Each user tries to offload its computation, accordingly with the available wireless bandwidth to reduce the energy consumption. Another offloading approach for multi-user was presented by Songtao Guo et al. [33], this work minimizes an offloading cost defined as a combination of energy consumption and processing time. The offloading policy decides which computation can be offloaded and allocates the wireless bandwidth and the processor frequency to each offloaded computation. In similar way, Keke Gai et al. [34] proposed a scheduler to assign the tasks between the local mobile terminals and the cloud in order to save energy consumption. Yuyi Mao et al. [35] presented an offloading policy that tries to offload the computation in multi-user scenario to an edge server, edge server. The proposed policy allocates the CPU frequency and the bandwidth, to each user, in order to reduce the energy consumption of the mobile terminal. Lastly, Feng Wang et al. [36] considered a multi-antenna AP, where each user tries to offload its computation to the edge server located in the AP. The objective is to design an energy-efficient offloading policy that decides which user offload, allocates thee bandwidth and the antenna used by each user that offload. Although

all these policies improve the performances of the mobile terminal, they rely on an unlimited capacity of the cloud. Consequently, they need some enhancements to be applied for the MEC where edge servers have limited computing resources.

The second category of works explores multi-user multiple remote server scenario. Tuyen X. et al. [37] consider a multi-edge server multi-user MEC. Every user has one computation task. The main objective is to decide which users should offload and to which edge server. They focused on partial offloading, where the offloading policy determines at runtime which computation must be performed locally and which one must be offloaded to a remote server. Yucen Nan et al. [38, 39], and Chongyu Zhou et al. [40] proposed computation offloading policies to reduce the energy consumption of fog nodes. They introduced an offloading policy where the fog nodes try to offload their computation to the remote cloud. For each fog node, the policy decides which computation must be offloaded to the remote cloud and which one must be performed locally by the node. In [39], the offloading policy has been extended to reduce the completion time of the applications. Similarly, Chongyu Zhou et al. introduced an online offloading policy. It can select the computations that should be performed by the nearest edge server in order to minimize a system-wide utility function, related to the execution time. These policies reduce the energy consumption of the fog server. But, they consider that the IoT device has a tiny computing capacity that cannot perform all possible applications. However, the mobile device has a considerable computing capacity, which can perform complex applications.

### 2.3.4 Application partitioning problem

The presented approaches assume always that the application is composed of one task, which can be offloaded or performed locally. However, in real work most application of mobile terminals are multitasking. For this aime, recent works explore the importance of computation offloading to improve the performance of the multitasking application. Dong Huang et al. [41] present an offloading approach in order to minimize the energy consumption of the user terminal. It decides which tasks should be performed locally and which ones should be offloaded to the remote cloud. Similarly, Thinh Quang Dinh et al. [42] propose a joint multitasking offloading and bandwidth allocation in MEC, with the objective of minimizing the energy consumption of the user terminal. Finally, Weiwei Chen et al. [43] study the device to device computation offloading, where user

terminals can offload their computation to each other. The objective is to offload some tasks from overloaded terminals to underloaded ones in order to reduce the total energy consumption. However, all these works assume that the tasks are independent and can be executed without any constrained relations among them. Unfortunately, this is not the case for the majority of the applications [44–46].

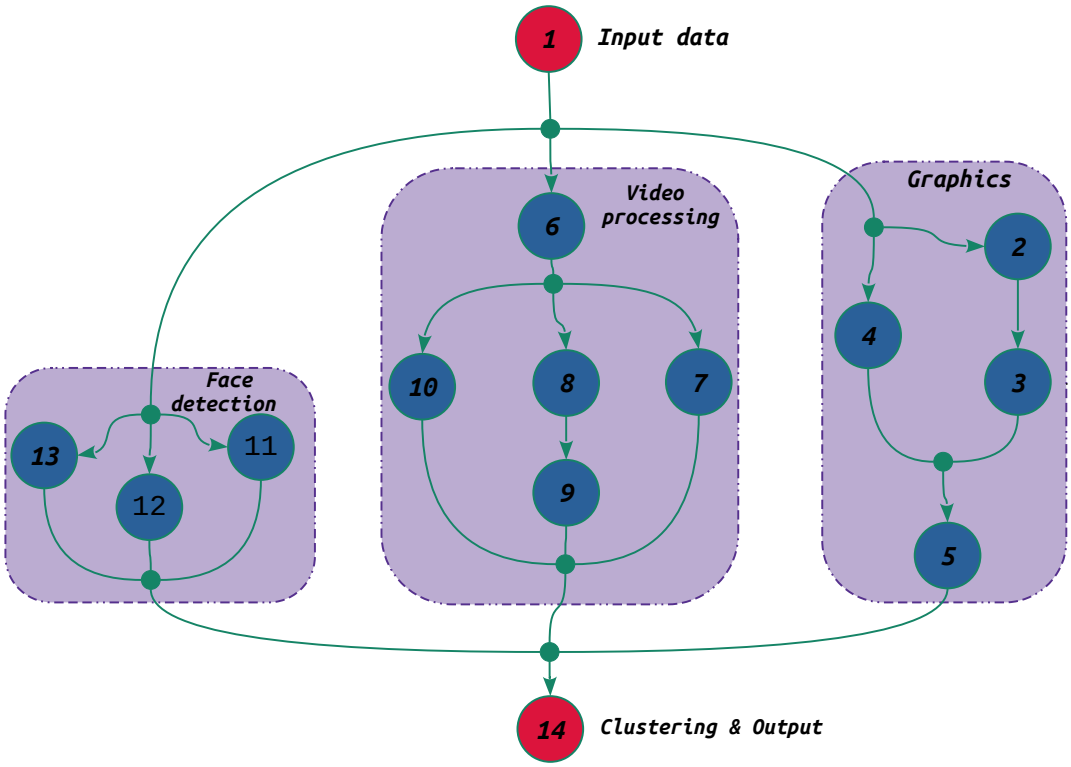



**Figure 2.4:** *An example of real-world multitasking mobile application 'video navigation application' as shown in [47].*

To address this issue, some works consider more realistic multitasking application and includes the precedence constraints between tasks. One of the most general model to define dependent multitasking application is to use a Directed Acyclic Graph (DAG) model. Using this model, we can represent the application as a set of tasks under precedence constraints and dependencies between tasks. Figure. 2.4 illustrates an example of a real-world multitasking application for Android terminal as shown in [47]. This application allows the user to navigate a video in order to recognize the people filmed in the video. We note that it is divided in components, where each one is composed of a set of dependent tasks. In [48] more real-world applications are presented. Considering this model, both Dong Huang et al. [41] and Songtao Guo et al. [33] have proposed a new offloading method that takes into account the dependencies between

the application's tasks. In addition, of selecting the tasks to be offloaded, Songtao Guo et al. [33] compute the bandwidth that should be allocated to each offloaded task. Moreover, Lei Yang et al. [49] focus only on applications with linear tasks graph and select the tasks to offload in order to minimize the terminal's energy consumption. Finally, Vincenzo De Maio et al. [50] explored the task's graph partitioning for offloading to MCC. They focus on the user preferences in the application partitioning to minimize the terminal's energy. The main drawback of the last approaches is related to the fact that a user can offload its tasks to only one edge server. As a result, it is difficult to maximize the parallelization of tasks to reduce the completion time.

In addition to the works referred above, the literature is also rich of works on graph-based tasks scheduling on multiprocessing systems, such as Yusuf Özkaya et al. [51] and Guan Wang et al. [52]. These works focus on the scheduling of multitasking applications on homogeneous and heterogeneous multiprocessor systems. The main objective is to minimize the completion time of the application. Unfortunately, even though these algorithms can be very efficient in a multiprocessor system, they can not be directly considered in MEC. The processor access delay on multiprocessor systems can be neglected compared to the time required to offload a task to the edge server. Indeed, the bandwidth constraints between the mobile terminal and the edge servers have a great impact on the decision and the location of the offloading.

### 2.3.5 Computation offloading platforms

In order to have real experiments of computation offloading in MCC/MEC, several works design offloading platforms for mobile terminals applications. A computation offloading platform intends to decrease energy consumption and reduce completion time of the application as well as to minimize the developer interference. Most of the existing platforms are composed of two entities as illustrated in Figure 2.5. Firstly, the client side performed by the mobile terminal, its role is to ensure the offloading on the terminal. For this purpose, it decides which tasks should be offloading and monitors the device and network conditions to determine when the offloading is beneficial. In addition, it implements communication protocol to exchange the data with the server side on the MEC. Secondly, the server side performed by the remote VM in an edge server, it ensures the processing of the offloaded tasks to the edge server. To this aim, it handles the execution of the tasks, monitors the network and the edge server information. It

also implement the communication protocol to communicate with the client side.

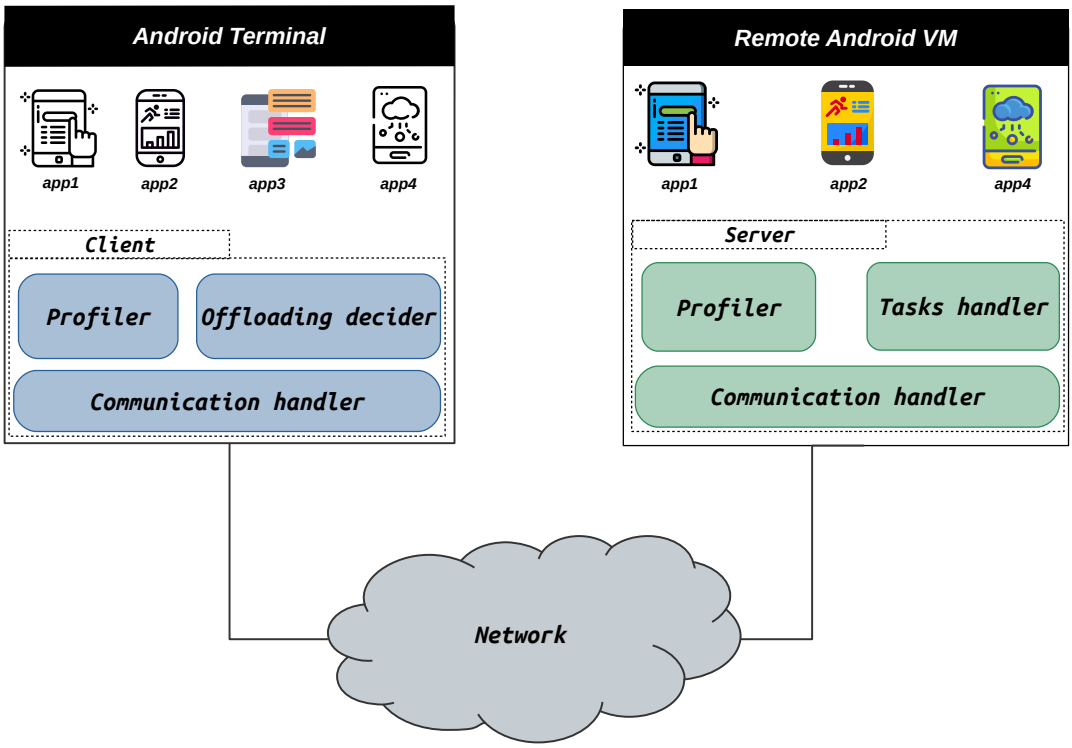



**Figure 2.5:** *Overview of Offloading platform.*

According to ESTI standardization an offloading platform has at least the following modules:

- **`Offloading Decider:`** is responsible to decide if the offloading is beneficial and which tasks should be offloaded. It uses the the data collected by Profiler to take these decisions.

- **`Task handler:`** is responsible of the execution of the offloaded tasks to the edge server. It receives those tasks from mobile terminals, then schedules their processing. At the end of the execution of each task, this module returns back the output to appropriate mobile terminal.

- **`Communication handler:`** is responsible to ensure the communication between the client side and the server side of the platform. It implements a network communication protocol such as Socket or Web Services APi.

- **`Profiler:`** is responsible to gather the information needed to evaluate the offloading. It collects three types of information:

- **`Application profiler:`** the number of calls of each method as well as the number of CPU cycles required for implementation are the important parameters in the measurement of energy consumption which is calculated in this section.
- **`Device profiler:`** that is related to the measurement of energy consumption of the mobile terminal on the basis of CPU cycles, Battery usage.
- **`Network profiler:`** used to profile the wireless networking environment to maximize energy savings. Therefore, a network measurement tool is applied to measure the round trip time and bandwidth.

Eduardo Cuervo et al. [53] implement Maui platform for Windows Phone terminals. This platform uses interface and annotations programming paradigm [54] to indicate which task to offload. Byung-Gon Chun et al. [55] propose offloading platform named Clonecloud, which uses Android Virtual Machine to offload tasks on the remote cloud. Clonecloud uses threads-based offloading strategy to offload tasks in order to reduce the completion time of the application. Similarly, Jose Benedetto et al. [46] design MobiCOP, which is an Android computation offloading platform based on Google cloud computing platform. Unfortunately, all the platforms described above have been designed to make the offloading possible but not to choose where to offload. In addition, they do not consider offloading to a multi-edge server MEC environment. In this thesis, we developed a new computation offloading platform dedicated to multi-edge server MEC environment. Using this platform, we set up a real experimentation in order to validate our offloading policy.

### 2.3.6 Discussions

The offloading works presented above have been focusing on reducing the offloading cost, energy consumption or completion time of the application. The offloading is always done to a predetermined remote server, in MCC or in MEC. The selection of this remote server is performed statically at deployment time based on end-user's density on the network; however, the density of users can change dynamically over time. Moreover, in MEC environment the computing capacity of every edge server is limited and cannot perform all the offloaded tasks. To avoid this situation, the most adopted strategy in the literature was to consider multi-level offloading approaches. Basically, the tasks are offloaded to the nearest edge server and this edge server offloads some tasks to another

one or the remote cloud when it is overloaded. Although multi-level offloading policies can improve the performances of the computation offloading approaches, they engender an additional overhead. In addition, in the multi-edge server scenario, where many edge servers are available around the mobile terminal, selecting always the same edge server is not the best strategy. Therefore, in this thesis, we propose a new offloading policy to improve the efficiency of computation offloading in MEC. The new policy must consider many edge servers for which a mobile terminal can offload its tasks, and compute optimal computation placements to optimize the offloading cost.

Moreover, the multitasking offloading approaches ignore the importance of the parallel offloading where multiple edge servers are available around the mobile terminal. As a consequence, in this thesis, we will explore the impact of the task-dependency graph on the computation offloading.

## 2.4 Conclusion

In this chapter, we have seen the required background for the understanding of the benefits, the challenges and the issues of computation offloading in MEC. One main benefit of the computation offloading in MEC is to enhance and enrich the performance of poor-resource devices such as mobile terminals. Several works investigated these issues and introduced many offloading policies and algorithms. These works improve the performances of the applications in MEC. However, most of this works ignored the affect of the dynamic edge servers selection and the application architecture on the performances of the offloading. Moreover, they do not explore the impact of the performance metrics on the offloading decision and efficiency. In addition, theoretical results are not consolidated with real experiments.

To deal with this problems, in this thesis, we will focus on computation offloading in MEC environment. Our works propose efficient offloading policies that answers the above notes, and provide a new computation offloading study that identifies all the factors that affect the offloading. In the next chapter, we will start by explored the importance of the edge server selection and the performance metrics.

CHAPTER 3

# Dynamic edge-server selection in MEC

*DM2-ECOP computation offloading policy*

**Abstract**

In this chapter, we will study the computation offloading decision and edge servers selection in multi-user multi-edge server MEC. The main objective is to propose an efficient offloading policy, which improves the performance of the application in such an environment. In addition, we will study the effect of the application on the offloading decision, and so the efficiency of the edge server selection. The results show that our proposal enhances the performance of the existing offloading approaches [56].

[56] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "DM2-ECOP: An Efficient Computation Offloading Policy for Multi-user Multi-cloudlet Mobile Edge Computing Environment". In: *ACM Transactions on Internet Technology.* 19.2 (2019), 24:1–24:24. DOI: 10.1145/3241666

## 3.1 Introduction

MEC emerges as a main paradigm to reduce the network delay for latency-sensitive applications, such as: mobile gaming, augmented-reality, face and speech recognition [6, 7], etc. In a real implementation of MEC, many edge servers are deployed with the existing APs, in order to support the high number of users that need to offload their computations. In such a scenario, both of MEC computational resources and wireless bandwidth are shared between the users. Consequently, the performance of the computation offloading depends strongly on the computing and wireless resources allocation strategies [31–33].

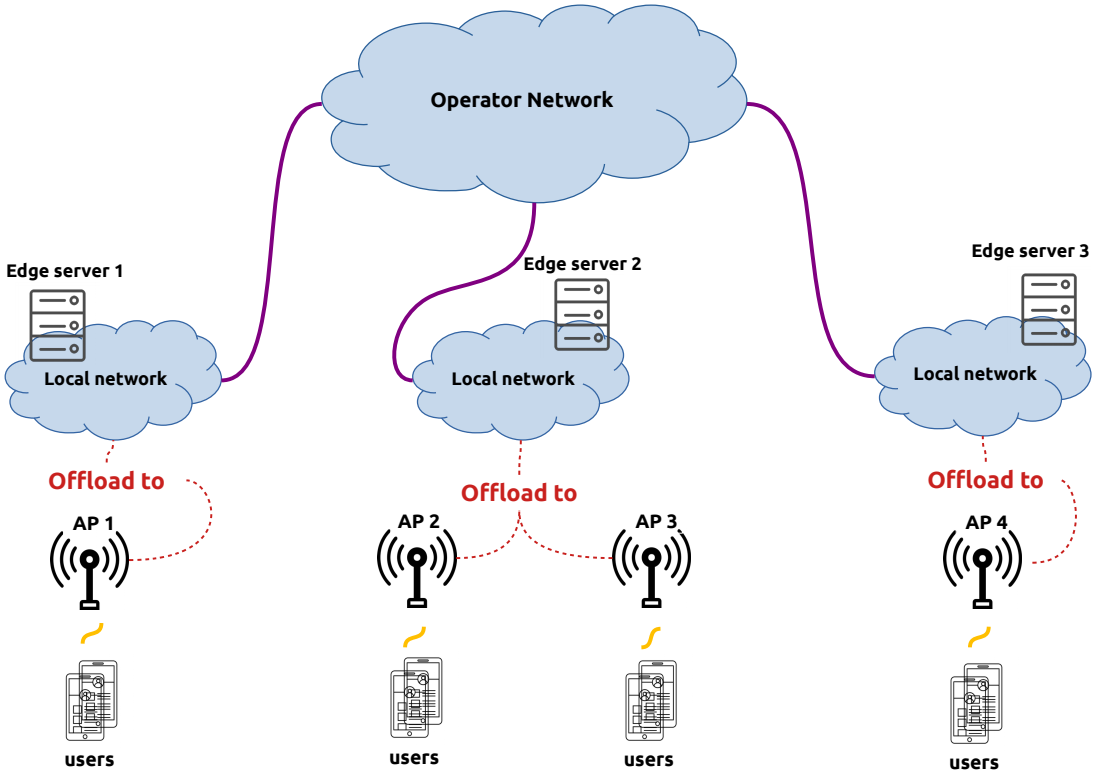



**Figure 3.1:** *An example of multi-user computation offloading in multi-edge server MEC environment of 4 APs and 3 edge servers.*

Recently, several works have investigated computation offloading for MEC environment [20, 21, 25]. Most of the proposed offloading policies rely on the bandwidth allocation, the computing resources allocation is done always in a predetermined edge-server. This edge-server is assigned statically based on users' density in the network [25, 26]. Therefore, as shown in Figure 3.1, users within a region will always offload to the same edge-server. Nevertheless, the dynamic density of users may imbalance the load between the edge-servers leading to suboptimal MEC capacities usage and longer

offloading delays. Therefore, to achieve high performance, an offloading policy must jointly consider bandwidth allocation, computation resource allocation, and edge-server selection.

In this chapter, we explore computation offloading in multi-user multi-edge server MEC. Our aim is to provide an efficient offloading policy, which determines the best offloading decision and edge servers selection for each user with the goal of reducing the total offloading cost. We present a new computation offloading policy named Distributed Multi-user Multi-edge server Efficient Computation Offloading Policy (DM2-ECOP), which targets to improve the performance of offloading in MEC environment. As in previous works [31–33], we assume that each user executes one and only one application at a time. However, unlike those works, we define two categories of applications that can be supported by the MEC: 1) *applications that must be performed remotely* and 2) *applications that can be performed either locally or remotely*, accordingly with the conditions at execution time.

DM2-ECOP formulates our computation offloading problem as a Mixed Binary Programming, and solves it using a distributed linear relaxation-based heuristic that follows the Lagrangian decomposition approach. It consists of two offloading decision levels:

1. **`The local decision Level:`** for each AP, the users associated with it are handled by a *local offloading manager*, in order to solve the offloading subproblem of this AP. Every subproblem is solved by GBC-SFH, Greedy Best edge server (Cloudlet) Selection First Heuristic, that determines which users should offload theirs tasks and select an edge server to each offloaded task.

2. **`The global decision level`**: a *global offloading manager* ensures that the offloading solution given by the local decision level complies with the edge servers resource constraints..

The remainder of the chapter is organized as the following. Section 3.2 presents the MEC environment infrastructure and system configurations. Section 3.3 details the computation offloading on multi-user multi-edge server MEC environment problem and the formulation of it. Our offloading policy is discussed in the section 3.4. The performances are illustrated, analyzed and discussed in section 3.5. The last section 3.6 draws a conclusion.

## 3.2 MEC system description and modeling

In this section, we describe the MEC system model and infrastructure. Then, we present the communication and computation offloading models. At the end, we introduce the offloading cost and the objective function of computation offloading problem in multi-user multi-edge server MEC environment.

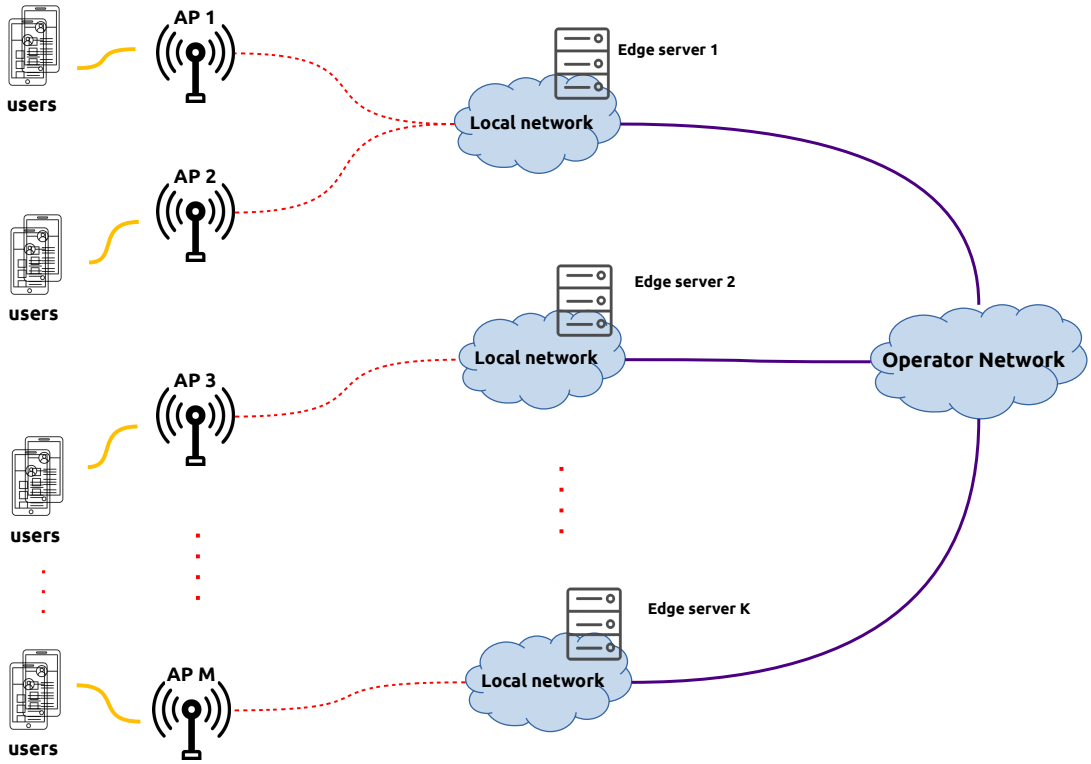



**Figure 3.2:** *An example of multi-user multi-edge server MEC environment composed of $M$ APs and $K$ edge servers.*

### 3.2.1 MEC infrastructure environment

Let us consider a MEC environment as illustrated in Figure 3.2. The infrastructure is composed of $M$ APs and $K$ edge servers deployed in the AP, and we suppose that the number of edge servers is less than the number of APs ($K \leq M$) [19]. In the following, we denote the set of APs by $\mathcal{M} = \{1, 2, ..., M\}$, and $\mathcal{K} = \{1, 2, ..., K\}$ as the set of the edge servers. Each edge server $k \in \mathcal{K}$ has a computational resources characterized by a fixed capacity, denoted $F^k$, of computing units. A computing unit is expressed in Giga-CPU Cycles and is defined as the number of CPU cycles per second available to perform the tasks in that edge server. We denote by $c^k$ the number of cycles per second allocated by server $k$ to perform any given offloaded task. Similarly to [32, 33], we

consider that $c^k$, $\forall k \in \mathcal{K}$, is fixed and does not change during the computation time. In addition, the computing capacity of any edge server is limited and can perform a limited number of tasks at a given time. Formally, $\forall k \in \{1, 2, ..., K\}$, $F^k < \infty$.

### 3.2.2 Users requirements

As illustrated in Figure 3.2, the MEC system is observed at a given time. Extension to continuous observation time is possible by discretizing the time into contiguous observation intervals. At a given time, we assume that the each AP $m$ is associated with $N_m$ users. Let consider $\mathcal{N}_m = \{1, 2, ..., N_m\}$ as the set of users associated with the $m^{th}$ AP. Let $(m, n)$ denote the $n^{th}$ user associated with the $m^{th}$ AP. Every user can communicate with the edge servers through the AP associated with. At the observation time, each user $(m, n)$ have exactly one application on his mobile terminal [18, 19, 35]. Note that our model can be extended to multiple applications easily by considering that each application is running by a virtual mobile terminal. The application is characterized by the following parameters:

1. The computational resource requirement in terms of CPU cycles, denoted by $\gamma_{m,n}$.
2. The amount of data uploaded to MEC, denoted by $up_{m,n}$.
3. The amount of data that must be downloaded by the user from MEC at the end of execution on the MEC, denoted by $dw_{m,n}$.
4. The maximum tolerated delay according to the Quality of Service (QoS) required by the application, denoted by $t_{m,n}$.

### 3.2.3 Tasks requirements

Inspired from previous works in computation offloading and mobile cloud computing [4, 57], we distinguish between two categories of computation offloading applications. This classification is based on when the offloading decision is token. The categories are:

1. The *static offloading decision task*
2. The *dynamic offloading decision task*

In the first category, the application is partitioned in advance, at the design time, between: a local part (task) that should always be executed in the mobile and a remote

part (task) that should always be executed remotely. As illustrated in Figure 3.3, the task's source code is already in the edge server, so the mobile terminal needs to transmit only the input data to that edge server. A typical example of static offloading decision task is the FLUID application on Android [58] which is used for particle simulations. In this case, a lightweight client is executed on the mobile terminal, and a more complex server part must be performed remotely in MEC, since it requires a high performance GPU computing processors that are not commonly available in mobile terminals.

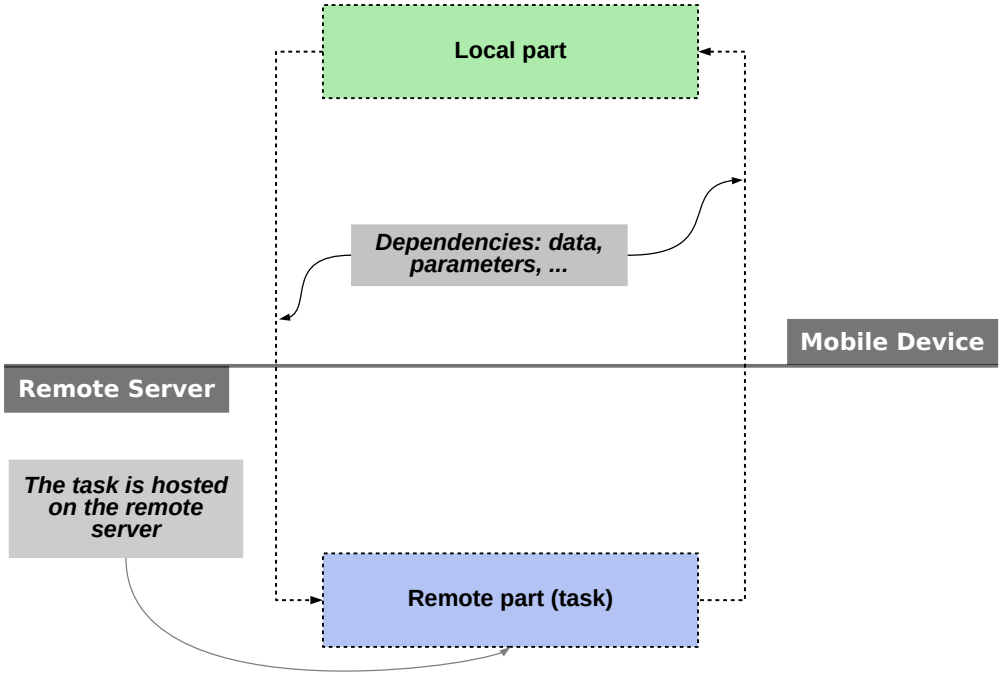



**Figure 3.3:** *Illustration of Static offloading decision task category of applications.*

In the second category, the application needs to be partitioned at runtime accordingly with the network and MEC resources availability. Basically, the mobile terminal needs to decide if it is useful to execute the task on the mobile terminal or to offload all or part of the task. In this case, as illustrated in Figure 3.4, the mobile terminal must transmit the source code and the input data of the task when the later is offloaded. An example of this kind of application is the Linpack benchmarks [59] for Android, which aims to measure the performances of Android terminals. This application can be either totally executed on the mobile terminal or partially offloaded to a edge server.

To simplify the analysis, we model both *static offloading decision task* and *dynamic offloading decision task* as a task with an offloading computation ratio noted as $a_{m,n}$.

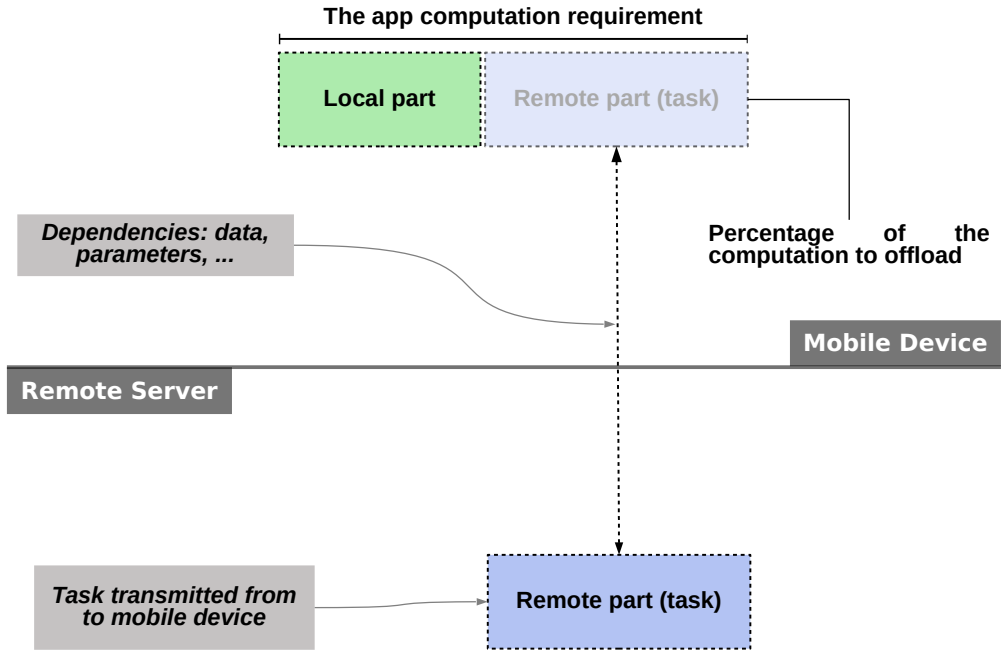



**Figure 3.4:** *Illustration of Dynamic offloading decision task category of applications.*

Basically, $a_{m,n}$ denotes the computation ratio of the application that should be executed remotely. In order to distinguish between *static offloading decision task* and *dynamic offloading decision task*, we consider that for *static offloading decision task* the offloading ratio is always equal to 1, which means that the tasks that belong to that category are always offloaded. The local part is considered as equal to zero since it will not affect the performances of the system since it should be always executed by the mobile terminal.

On the other hand, for *dynamic offloading decision task*, this offloading computation ratio can take any value between 0 and 1 (i.e $a_{m,n} \in [0,1]$). In addition, for simplicity, but without the loss of generality, we also assume that when a task is offloaded with a given offloading computation ratio, then the amount of data that should be transmitted are also proportional to that ratio. However, the output of the task, noted as $dw_{m,n}$, does not change whatever the value of $a_{m,n}$.

In order to indicate to which category the application of user $(m,n)$ belongs, we introduce the binary variable $y_{m,n}$, which is equal 1 for *static offloading decision task*s and 0 for *dynamic offloading decision task*.

Finally, due to hardware and software constraints required by the task, we assume that some edge servers cannot perform some tasks. We define another binary variable,

$g^k_{m,n}$, to indicate if the edge server $k$ can perform the task. Thus $g^k_{m,n}$ is equal to 1 when the $k^{th}$ edge server can perform the task of user $(m,n)$, 0 otherwise.

### 3.2.4 The local processing time

The local processing time of a task depends to its category. For static offload decision task, the local processing time is equal zero (0), since, by definition, it must always be offloaded. However, for dynamic offload decision task the processing time relies on the percentage $a_{m,n}$ of the offloaded computation. Thus, the remaining computation must be performed locally by the user's mobile terminal. At the observation time, we assume that the user's terminal has a local computing capacity, denoted as $f_{m,n}$, used to performed the task locally. Consequently, the local processing time, noted by $T^l_{m,n}$, can be estimated by the ratio between the non offloaded computation and the user's terminal capacity. It can be given as follows:

$$T^l_{m,n} = (1 - a_{m,n}) \cdot \frac{\gamma_{m,n}}{f_{m,n}} \tag{3.1}$$

From the above equation, we can notice that the local processing time of a Static offload decision task is equal to zero ($T^l_{m,n} = 0$) since $a_{m,n}$ for static offload decision task is always equal to 1.

### 3.2.5 The remote processing time

The remote execution of the task of user $(m,n)$ in the edge server $k$ is composed of three main steps: i) the upload time, ii) the processing time and iii) the download time. These three times compose the offloading time. Each one of them can be expressed as in the following.

- **`The upload time`**, denoted by $T^{up,k}_{m,n}$, is the time required for the edge server $k$ to receive all the input data of the task $(m,n)$. More precisely, it is the time to upload the amount of data $a_{m,n} \cdot up_{m,n}$ to the edge server $k$. $T^{up,k}_{m,n}$ can be be written by the following equation:

$$T^{up,k}_{m,n} = T^{tx}_{m,n} + T^{bh,k}_{m,n} + \mathcal{D}^k_m \tag{3.2}$$

Where $T^{tx}_{m,n}$ is the access time and represents the time to upload the input data to

the AP according to the available wireless bandwidth. It can be given as follows:

$$T_{m,n}^{tx} = a_{m,n} \cdot \frac{up_{m,n}}{w_{m,n}^{tx}} \tag{3.3}$$

where $w_{m,n}^{tx}$ is the upload wireless bandwidth allocated to the user $(m,n)$ at the AP $m$ as detailed in section 3.2.6. $T_{m,n}^{bh,k}$ is the backhaul time, and it represents the time to upload the input data between the AP and the target edge server. It can be expressed aw following:

$$T_{m,n}^{bh,k} = a_{m,n} \cdot \frac{up_{m,n}}{W_m^{bh,k}} \tag{3.4}$$

where $W_m^{bh,k}$ refers to the end to end backhaul bandwidth available between the AP $m$ and the edge server $k$. And, $\mathcal{D}_m^k$ is the latency between the AP $m$ and the edge server $k$.

- **`The processing time`**, denoted by $T_{m,n}^{e,k}$, is the time required to execute the task $(m,n)$ in the edge server $k$. Similar to the local processing time in equation 3.1, $T_{m,n}^{e,k}$ can be expressed by the following equation:

$$T_{m,n}^{e,k} = a_{m,n} \cdot \frac{\gamma_{m,n}}{c^k} \tag{3.5}$$

- **`The download time`**, denoted by $T_m^{dw,k}$, represents the time needed for mobile terminal to receive the task's output $dw_{m,n}$ from the edge server $k$. From above the assumptions , $T_{m,n}^{k,dw}$ can be written as follows:

$$T_{m,n}^{dw,k} = T_{m,n}^{rx} + T_{m,n}^{k,bh} + \mathcal{D}_m^k \tag{3.6}$$

where $T_{m,n}^{rx}$ is the time required to download the task's output from the AP to the mobile terminal. It can be expressed as:

$$T_{m,n}^{rx} = \frac{dw_{m,n}}{w_{m,n}^{rx}} \tag{3.7}$$

where $w_{m,n}^{rx}$ is the download bandwidth available between the AP and the mobile terminal. $T_{m,n}^{k,bh}$ is the time to download the task's output from the edge server $k$

to the AP $m$ through the backhaul network. Similar to $T^{bh,k}_{m,n}$, it is given by:

$$T^{k,bh}_{m,n} = \frac{dw_{m,n}}{W^{bh,k}_{m}} \tag{3.8}$$

Finlay, the total remote processing time of a task $(m,n)$ on edge server $k$, denoted by $T^{r,k}_{m,n}$, can be expressed by the following equation:

$$T^{r,k}_{m,n} = T^{up,k}_{m,n} + T^{e,k}_{m,n} + T^{dw,k}_{m,n} \tag{3.9}$$

### 3.2.6 Shared wireless access bandwidth

Let now focus on the allocated wireless bandwidth at AP for each user. To develop the expression of the upload and download allocated bandwidth for every user that offloads its tasks to MEC, we suppose that the AP uses an orthogonal frequency division multiple-access (OFDMA) technology. Without loss of generality and in a seek of presentation simplicity, we assume that the bandwidth allocated on the uplink (i.e. from the user to the AP) is the same to the bandwidth allocated on the downlink (i.e. from the AP to the user). In addition, we admit that the channel from the user to the AP follows quasi-static Rayleigh fading. In other words, the channel state is stationary during the offloading period of the applications of all users.

Let $B_m$ denote the available wireless bandwidth in the AP $m$. This quantity is expressed in Hertz (Hz). Since we are considering OFDMA, $B_m$ is divided into $\mathcal{B}_m$ subcarriers. For every user $(m,n)$, we define $p^{tx}_{m,n}$ as the user terminal's transmission power, and $h^{tx}_{m,n}$ as the power gain. We consider both path loss and shadowing attenuation as in [60]. We model the noise as *Additive White Gaussian Noise (AWGN)* [60] random variable with zero mean and variance $\sigma^2_m$. Thus, the maximum achievable data rate (in bps) for the uplink can be expressed as follows:

$$w^{tx}_{m,n} = \upsilon_{m,n} \cdot \frac{B_m}{\mathcal{B}_m} \cdot log_2(1 + \frac{p^{tx}_{m,n} \cdot (h^{tx}_{m,n})^2}{\Gamma(b^{tx}_{m,n}) \cdot d^2_{m,n} \cdot \sigma^2_m}) \tag{3.10}$$

where $\Gamma(BER)$ represents the SNR margin introduced to meet the desired target bit error rate (BER) with a QAM constellation, $b^{tx}_{m,n}$ is the BER of user $(m,n)$, $d_{m,n}$ is the distance between the user and the AP, and $\upsilon_{m,n}$ the number of subcarriers allocated to

the user. $\Gamma(BER)$ can be expressed as follows:

$$\Gamma(x) = -2 \cdot \frac{log(5x)}{3} \tag{3.11}$$

Similarly, the maximum achievable data rate (in bps) for downlink can be derived by the following equation:

$$w_{m,n}^{rx} = \upsilon_{m,n} \cdot \frac{B_m}{\mathcal{B}_m} \cdot log_2(1 + \frac{p_m^{ap} \cdot (h_{m,n}^{rx})^2}{\Gamma(b_{m,n}^{rx}) \cdot d_{m,n}^2 \cdot \sigma_m^2}) \tag{3.12}$$

where $p_m^{ap}$ denotes the transmission power of the AP. $h_{m,n}^{rx}$ indicates the power gain from the user to the AP when user receives data due to path loss and shadowing attenuation [60].

Finally, regarding the backhaul network, the bandwidth capacity between any AP $m \in \mathcal{M}$ and any edge server $k \in \mathcal{K}$, denoted $W_m^{bh,k}$ and $W_m^{k,bh}$, is supposed to be symmetric and fixed during the application execution period.

### 3.2.7 The offloading cost

In this section, we focus on defining the offloading cost model. We define the offloading cost as a combination of the energy consumption and the execution time of the application. In the following, we present both local offloading cost and remote offloading costs.

- **`Local offloading cost:`** According to the last considerations the local offloading cost is expressed as a combination of the energy consumption and the execution time of the tasks, as following:

$$\mathcal{Z}_{m,n}^l = \beta_{m,n} \cdot E_{m,n}^l + (1 - \beta_{m,n}) \cdot T_{m,n}^l \tag{3.13}$$

  where $E_{m,n}^l$ and $T_{m,n}^l$ are respectively the total amount of energy and processing time of the local part of the application of user $(m,n)$. $\beta_{m,n}$ is a weighting parameter between execution time and energy consumption of the user's offloading decision. When the battery of user's mobile terminal is at a low state, the user needs to reduce the energy consumption then it can set $\beta_{m,n} = 1$. On the other hand, when a delay sensitive task is running the user can set $\beta_{m,n} = 0$ to give more priority to the execution time. To get a more flexible cost model, we allow

multi-criteria offloading policy by considering energy consumption, execution time or a combination of both of them.

Considering the energy consumption model, we use the model proposed in [61, 62]. Using this model, we can compute $E^l_{m,n}$ as following:

$$E^l_{m,n} = \kappa_{m,n} \cdot (f_{m,n})^3 \cdot T^l_{m,n} \tag{3.14}$$

where $\kappa_{m,n}$ is the effective switched capacitance, which depends on the chip architecture and is used to adjust the processor frequency. In the following, we set $\kappa_{m,n} = 10^{-9}$ as shown in [61, 62].

- **`Remote offloading cost:`** The total amount of energy consumed by the user's mobile terminal to perform the task remotely is equal to the energy used when the terminal turns on the radio in the transmission mode to send the data to the remote server, plus the energy used when the device turns the radio in idle mode to wait the task completion, plus the energy used by the device when it turn again the radio in the reception mode in order to receive the result data from the remote server. This consumed energy can be expressed as following:

$$E^{e,k}_{m,n} = p^{tx}_{m,n} \cdot T^{tx}_{m,n} + p^{idle}_{m,n} \cdot (T^{e,k}_{m,n} + T^{bh,k}_{m,n} + T^{k,bh}_{m,n} + 2 \cdot \mathcal{D}^k_m) + p^{rx}_{m,n} \cdot T^{rx}_{m,n} \tag{3.15}$$

where $p^{tx}_{m,n}$, $p^{rx}_{m,n}$ is the power consumption when the radio interface is set to transmission or reception mode, and $p^{idle}_{m,n}$ is the power consumption in the case when the radio interface is set to of idle mode [61–63]. Finally, we can define the remote offloading costs as following:

$$\mathcal{Z}^{e,k}_{m,n} = \beta_{m,n} \cdot E^{e,k}_{m,n} + (1 - \beta_{m,n}) \cdot T^{r,k}_{m,n} \tag{3.16}$$

## 3.3 Computation offloading problem formulation and decomposition

In order to propose an efficient offloading policy, we formulate the problem as an optimization problem. Then, we use Lagrangian relaxation to decompose the problem into subproblems and solve each one separately.

### 3.3.1 The completion time constraint

Let $T_{m,n}$ denotes the total completion time of the task of user $(m,n)$. From the above system description and modeling section, one can see that $T_{m,n}$ depends on the location of the execution of the task. Precisely, it depends the the processing time and eventually the upload and download transmission times in case of offloading. Formally, if the task of user $(m,n)$ is performed locally, then $T_{m,n} = T^l_{m,n}$. Otherwise, if the task is offloaded on the edge server $k$ then $T_{m,n} = T^{r,k}_{m,n}$.

Hence, to integrate applications' Quality of Services requirements (QoS), we associate to each offloadable task a time constraint threshold. Precisely, we define a maximum completion time threshold, denoted $t_{m,n}$ to any task, $\forall (m,n) \in (\mathcal{M},\mathcal{N}_m)$. The threshold is obtained based on the characteristics of the application [32, 53]. It can be determined as the duration of the submission of the task until receiving the response for an interactive application [7]. Also, it is a hard constraint for any task, $\forall (m,n) \in (\mathcal{M},\mathcal{N}_m)$. The completion time can be expressed as follows:

$$T_{m,n} = (1 - \sum_{k=1}^{K} x^k_{m,n}) \cdot T^l_{m,n} + \sum_{k=1}^{K} x^k_{m,n} \cdot (T^{r,k}_{m,n})$$

In other words, the optimal offloading solution of the problem must satisfy the following constraint:

$$C3 : T_{m,n} < t_{m,n} \tag{3.17}$$

As stated before, the purpose of our offloading policy is to determine which tasks should be offloaded and to which edge server in order to satisfy the completion time constraint. In addition to this constraint, our purpose is to determine the optimal offloading policy which minimize the overall energy consumed by the users terminals. In the following sections we will detail energy consumption models for both, local and remote tasks processing.

### 3.3.2 Problem formulation

As introduced earlier, our objective is to propose an efficient offloading policy, and it decides the users that should offload their tasks, determines the amount of computation to offload and select an edge server for each user, while minimizing the total offloading cost. Let us denote by $x^k_{m,n}$ the offloading decision for the task of the user $(m,n)$ on the

edge server $k$, which mean that $x^k_{m,n} = 1$ if the user $(m,n)$ offloads its task to the edge server $k$, 0 otherwise. Given the system description and according to the QoS and edge servers' resources capabilities constraints, the problem can be formulated as follows:

$$
\begin{aligned}
&\textbf{Minimize} \sum_{m}^{M} \sum_{n}^{N_m} \mathcal{Z}_{m,n} \\
&\textbf{Subject to } C3 \textbf{ and :} \\
&C1 : \sum_{k=1}^{K} x^k_{m,n} \leq 1, \forall m \in \mathcal{M}, n \in \mathcal{N}_m \\
&C2 : y_{m,n} - \sum_{k=1}^{K} x^k_{m,n} \leq 0, \forall m \in \mathcal{M}, n \in \mathcal{N}_m \\
&C4 : x^k_{m,n} \leq g^k_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}_m, k \in \mathcal{K} \\
&C5 : \sum_{m}^{M} (\sum_{n}^{N_m} x^k_{m,n} \cdot c^k) \leq F^k, \forall k \in \mathcal{K} \\
&C6 : x^k_{m,n} \in \{0,1\}, \forall m \in \mathcal{M}, n \in \mathcal{N}_m, k \in \mathcal{K} \\
&C7 : a_{m,n} \in [0,1], \forall m \in \mathcal{M}, n \in \mathcal{N}_m \\
&C8 : a_{m,n} \geq y_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}_m
\end{aligned}
\tag{3.18}
$$

where $\mathcal{Z}_{m,n}$ is the offloading cost of the user's $(m,n)$ task. $\mathcal{Z}_{m,n}$ can be expressed by the following formula:

$$
\mathcal{Z}_{m,n} = (1 - \sum_{k=1}^{K} x^k_{m,n}) \cdot \mathcal{Z}^l_{m,n} + \sum_{k=1}^{K} x^k_{m,n} \cdot \mathcal{Z}^{e,k}_{m,n} \tag{3.19}
$$

As indicated in the problem formulation, our objective is to minimize the total offloading cost of the users of the network. The first constraint $(C1)$ ensures that each task is assigned at most to one edge server. Constraints $(C2)$ guarantee that any static offloading decision task must be assigned to exactly one edge server, and a dynamic offloading decision task may be assigned to at most one edge server. The next constraint $(C3)$ is the completion time constraint defined in 3.3.1. The next constraint $(C4)$ ensures that every offloaded task must be performed by an edge server that meets the hardware and software required by the task. The constraint $C5$ shows that it is not possible to exceed the computing capacity of each edge server. Constraint $C6$ ensures that any

decision variable is a binary variable. Finally, constraints ($C7$) indicates that the ratio of the offloaded computation must be a real value between 0 and 1. And, ($C8$) ensures that each static offloading decision task must always be entirely offloaded.

**Theorem 3.1.** *The problem 3.18 is nonlinear mixed problem with nonlinear objective function and constraints. It is NP-hard problem.*

*Proof.* Let us consider a special case where the same number of users are associated with each AP, and all tasks belong to static offloading decision tasks category. Consequently, all the tasks must be offloaded to the MEC. In addition, the bandwidth allocated to each user is known in advance, and all the variable $a_{m,n} = 1$ as assumed previously. Thus, this special case forms a linear binary problem. It can be easily reduced to the General Assignment Problem (GAP) with assignment restrictions, which is NP-hard as demonstrated by the work presented in [64]. Since the special case is NP-hard, the problem 3.18 is too. □

Considering the NP-hardness of the offloading optimization problem, it is difficult to achieve an optimal solution in a reasonable time. In the next subsections, we propose a simplification version of the problem (3.18) using Lagrangian relaxation and decomposition approach to get a good feasible solution quickly.

### 3.3.3 Problem decomposition

To solve the above problem, we need to use a heuristic that reduces its complexity. For this purpose, we call for the decomposition approach. Decomposing a complex optimization problem consists of breaking it up into smaller ones, called subproblems, and solving each of the smaller ones separately. Unfortunately, the constraint $C5$ is considered as a complicating constraint [65, 66], since it involves the local variables of more than one subproblem. Consequently, the decomposition of the problem (3.18) does not work in one step and the subproblem cannot be solved independently. For these kinds of complex problems, there are advanced decomposition techniques that solve the problem by iteratively solving a sequence of subproblems. In this chapter, we employ one of the most popular decomposition techniques, the Lagrangian relaxation [65, 66]. The idea of the Lagrangian relaxation comes up in the context of using Lagrangian multipliers to decompose the problem; thus, we introduce the Lagrangian multipliers $\lambda = [\lambda^k, k \in \mathcal{K}]^T$ on the constraint $C5$, where $\lambda^k$ denotes the price of all the tasks

performed by the $k^{th}$ edge server. $\mathcal{X}$ and $\mathcal{A}$ are the set of the offloading decision variables and the set of the offloading ratio, respectively. The Lagrangian function is given as follows:

$$L(\mathcal{X},\mathcal{A},\lambda) = \sum_{m}^{M}\sum_{n}^{N_m} \mathcal{Z}_{m,n} + \sum_{k}^{K} \lambda^k \sum_{m}^{M}\sum_{n}^{N_m} (x_{m,n}^k \cdot c^k - F^k)$$

$$= \sum_{m}^{M}\sum_{n}^{N_m} \mathcal{Z}_{m,n} + \sum_{m}^{M}\sum_{n}^{N_m}\sum_{k}^{K} \lambda^k x_{m,n}^k \cdot c^k - \sum_{k}^{K} F^k$$

$$= \sum_{m}^{M}\sum_{n}^{N_m} (\mathcal{Z}_{m,n} + \sum_{k}^{K} \lambda^k x_{m,n}^k \cdot c^k) - \sum_{k}^{K} \lambda^k F^k$$

The Lagrange dual problem for primal problems (3.18) is then given by:

$$\max_{\lambda} \min_{\mathcal{X},\mathcal{A}} L(\mathcal{X},\mathcal{A},\lambda)$$

We can see that the Lagrange dual problem is separable into two levels. Level 1 is the inner minimizing, which consists of $M$ subproblems each one concerns only one AP. Level 2 is outer maximization, which is the master problem that considers the global variables and constraint $C5$.

Focusing on this observation, we introduce a new offloading policy named Distributed Multi-user Multi-Edge Server Efficient Computation Offloading Policy (DM2-ECOP). In the following, we describe the proposed computation offloading policy

## 3.4 DM2-ECOP: Distributed Multi-user Multi-edge server Efficient Computation Offloading Policy

As introduced in the last section, the Lagrange dual is decomposable into $M$ subproblems. Each subproblem tries to find an optimal offloading decision and edge server selection to the users associated to one AP. Considering this characteristic, we design a new offloading policy DM2-ECOP. As shown in Figure 3.5, DM2-ECOP has two levels of decision. `The local offloading manager` is responsible for the offloading decision and edge server selection of an AP. It solves the associated subproblem, then sends the offloading decision and the edge server selection to the centralized decision level. `The MEC computation offloading manager` receives the solution of all subproblems, then it ensures that the obtained offloading solution is feasible and respects all the

constraints. After, it updates the Lagrangian multipliers and transmits the new values to every `local offloading manager` in order to improve the local solutions.

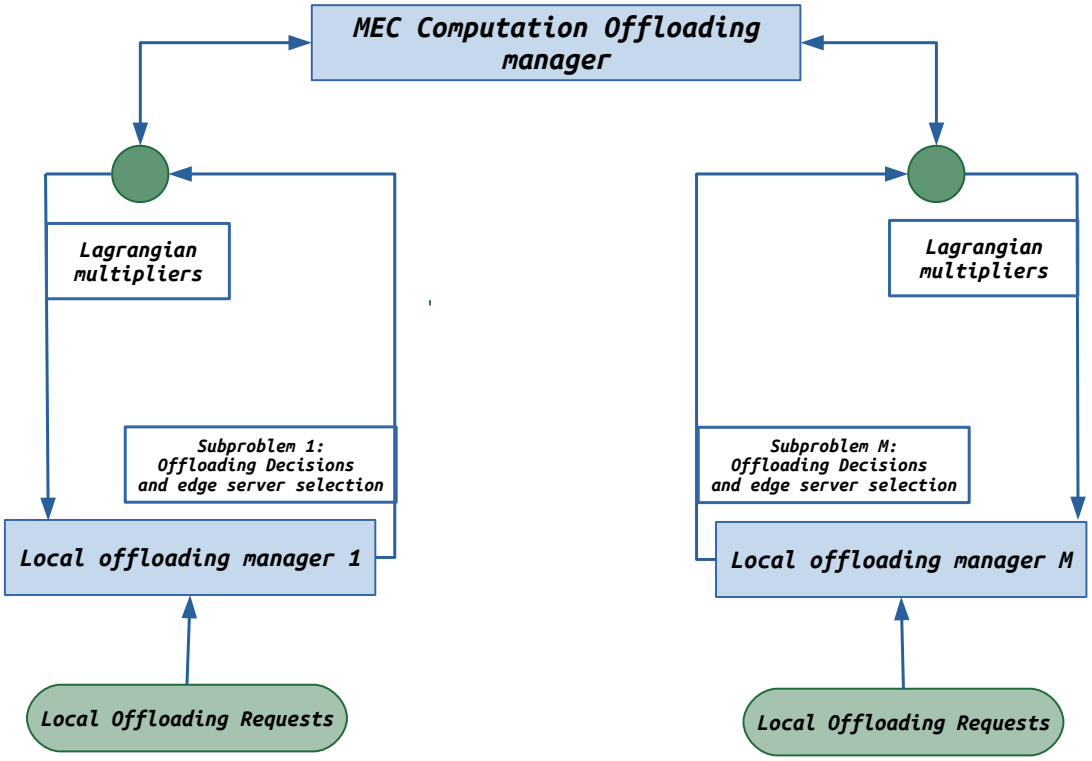



**Figure 3.5:** *DM2-ECOP offloading policy architecture overview.*

### 3.4.1 Local offloading manager

The local offloading manager tries to solve the subproblem of one AP, in order to decide which user can offload. Then, it selects the appropriate edge server to perform the task of each user. According to the previous considerations, we can formulate the subproblem of a local offloading manager as follows:

$$\textbf{Minimize} \sum_{n}^{N_m} (\mathcal{Z}_{m,n} + \sum_{k}^{K} \lambda^k x_{m,n}^k \cdot c^k)$$

**Subject to:**

$$\text{constraints C1 - C4 and C6-C8} \tag{3.20}$$

DM2-ECOP needs to know the bandwidth allocation in order to decide whether a user should offload its task or not. To overcome this dependency problem, we use a simple channel allocation model. Let define **the offloading capacity** of AP $m$, denoted

by $\pi_m$, which represents the number of tasks that are offloaded through AP $m \in \mathcal{M}$ to an edge server $k \in \mathcal{K}$. Obviously, $0 \leq \pi_m \leq N_m$. Formally **the offloading capacity** can be defined as the following.

**Definition 3.1.** *the **offloading capacity** of the AP $m$ is defined as the number of tasks that have been accepted to being performed by edge servers in the MEC environment. We note it by $\pi_m$, and it is given by:*

$$\pi_m = \sum_{n}^{N_m} \sum_{k}^{K} x_{m,n}^{k}$$

To measure the allocated bandwidth to upload and download the offloaded tasks at every AP, we suppose that the subcarriers are allocated with equity to all the users that offload. Consequently, the number of subcarriers allocated to each user is given by:

$$\upsilon_{m,n} = \frac{\mathcal{B}_m}{\pi_m} \tag{3.21}$$

The strategy of solving the subproblem consists to find the optimal value of the offloading capacity ($\pi_m$) that gives the minimal offloading cost. To this aim, we draw on a heuristic that iterates all the possible values of $\pi_m$ with the help of the constraint $C9$, presented below, to the subproblem (3.20).

$$C9 : \sum_{n}^{N_m} \sum_{k}^{K} x_{m,n}^{k} = \pi_m \tag{3.22}$$

To reach good offloading solution quickly, the local offloading manager uses a greedy heuristic to solve the subproblem ( 3.20) maned GBC-SFH, Greedy Best edge Server Selection First Heuristic. GBC-SFH decides which users should offload, determines the amount of the tasks' computation to offload and select the best edge server to perform each offloaded computations. As illustrated in algorithm 1, GBC-SFH iterates all the possible values of the offloading capacity, denoted $\Pi_m$, in an increasing order. Following the definition 3.1, $\Pi_m = \{0, ..., N_m\}$. For every possible value of $\pi_m$, GBC-SFH allocates the network bandwidth to each user using equation equations 3.10, 3.12 and 3.21. The idea of the heuristic is to add the constraint C9, defined in equation 3.22, to the subproblem in order to ensure that the offloading capacity is exactly equal to $\pi_m$ at each iteration. Firstly, it starts by assigning each one of the static offloading tasks category

to the best edge server $k$. This server is the one that minimizes the Lagrangian cost $\mathcal{Z}_{m,n}^{e,k} + \lambda^k \cdot c^k$ under the constraints $C1 - C4$ and $C6 - C8$.

**Algorithm 1** *the local offloading manager: pseudo code of GBC-SFH*

**Input:**
$\Pi_m$: Set of offloading capacity ;
**Output:** output the offloading decisions $\mathcal{X}$, ratios $\mathcal{A}$ and cost $\mathcal{Z}$;
Sort $\Pi_m$ in increasing order;
**for** $\pi_m \in \Pi_m$ **do**
  allocate bandwidth using equations 3.10, 3.12 and 3.21;
  offload each static offloading decision task to the edge server $k$ that minimizes $\mathcal{Z}_{m,n}^{e,k} + \lambda^k \cdot c^k$ under constraints $C1 - C4$ and $C6 - C8$;
  $nb_{offloaded\_task}$ = number of static offloading decision tasks;
  compute $\xi_{m,n}$ for every dynamic offloading decision task;
  Sort dynamic offloading decision tasks in decreasing order of $\xi_{m,n}$ ;
  **while** $nb_{offloaded\_task} \leq \pi_m$ **do**
    Select the edge server $k$ that minimizes $\mathcal{Z}_{m,n}^{e,k} + \lambda^k \cdot c^k$ under constraints $C1 - C4$, $C6 - C7$ and $a_{m,n} = 1$;
    Compute the optimal value of $a_{m,n}$ using theorem 3.2;
    **if** $a_{m,n} == 0$ **then**
      there is no offloading. This dynamic Offloading task must be performed locally;
    **else**
      Offload this dynamic Offloading task to the edge server $k$;
      $nb_{offloaded\_task} + +$;
    **end if**
    **if** (there is no more task) and $(nb_{offloaded\_task} < \pi_m)$ **then**
      break the while-loop. There is no feasible solution for this value of $\pi_m$;
    **end if**
  **end while**
  all the remaining tasks must be performed locally;
  update the best offloading cost $\mathcal{Z}$, ratio $\mathcal{A}$ and decision $\mathcal{X}$;
**end for**

For each task of the dynamic offloading decision tasks category, GBC-SFH tries to select the best edge server and compute the optimal ratio $a_{m,n}$ of the computation to offload. According to the resource availability, GBC-SFH can offload the task or perform it locally, $a_{m,n} = 0$, by the user's mobile terminal. Since the wireless bandwidth at the AP may not be enough to offload all the dynamic offloading decision task, we need to define an order to determine which dynamic offloading decision task is preferred for the offloading. To this purpose, we define an offloading priority for each task according to the following formula:

$$\xi_{m,n} = \mathcal{Z}_{m,n}^{l} - \min_{k \in \mathcal{K}}(\mathcal{Z}_{m,n}^{e,k}); \quad \text{under } a_{m,n} = 1 \tag{3.23}$$

here the offloading priority is the local cost minus the cost when all the computation is offloaded to the best edge server. The idea here is that where $\xi_{m,n}$ is going higher, the user $(m,n)$ is more preferred to offload its task. Unlike the static offloading decision tasks, for dynamic offloading decision tasks we need to compute $a_{m,n}$, to recall it defines the amount of the computation to offload. To this end, GBC-SFH uses two-step method. In the first step, it selects the best edge server to offload the computation of the user $(m,n)$. At this step, GBC-SFH chooses the edge server that minimizes the Lagrangian Cost $\mathcal{Z}^{e,k}_{m,n} + \lambda^k \cdot c^k$ under the constraints $C1 - C4$, $C6 - C7$ and $a_{m,n} = 1$. After the selection of the best edge server, GBC-SFH computes the optimal value of $a_{m,n}$ for the current user. The optimal value of $a_{m,n}$ is the solution of the following problem:

$$\begin{aligned} & min(\mathcal{Z}^{e,k}_{m,n} + \mathcal{Z}^{l}_{m,n}) \\ & \textbf{Subject to:} \quad \text{constraint } C7. \end{aligned} \tag{3.24}$$

the problem 3.24 is a simple problem with one variable. Its optimal solution can be achieved by derivative sign rules [67]. Theorem 3.2 shows when the minimum of this problem will be achieved.

**Theorem 3.2.** *Let us define $\psi_{m,n}$ and $\mu_{m,n}$ the upload data-computing ratio of the dynamic offloading decision task, and the local-remote offloading cost ratio of the user $(m,n)$, respectively. They are given as follows:*

$$\psi_{m,n} = \frac{up_{m,n}}{\gamma_{m,n}}$$

$$\mu_{m,n} = \frac{w^{tx}_{m,n} \cdot W^{bh,k}_{m}}{c^k \cdot f_{m,n}} \times \frac{([\beta_{m,n} \cdot p^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot f_{m,n} - [\beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^3 \cdot c^k] - [1 - \beta_{m,n}] \cdot c^k)}{([p^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}] \cdot W^{bh,k}_{m} + [\beta_{m,n} \cdot p^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot w^{tx}_{m,n})}$$

*The minimum of the problem 3.24 is achieved when:*

- $a_{m,n} = 1$, *if and only if :*

$$\psi_{m,n} < \mu_{m,n}$$

- $a_{m,n} = 0$, *if and only if:*

$$\psi_{m,n} > \mu_{m,n}$$

- *the problem is constant, if and only if :*

$$\psi_{m,n} = \mu_{m,n}$$

*Proof.* The proof of the theorem 3.2 is detailed in the appendix A.1. □

Using the theorem 3.2, we have three possible scenarios for dynamic offloading decision tasks: 1) when $a_{m,n} = 1$, the whole computation must be offloaded, 2) when $a_{m,n} = 0$, in this case there is no offloading and 3) when the problem 3.24 is constant, than all possible values of $a_{m,n}$ gives the same performance. As the computation offloading is a solution to improve the performance, we choose to not offload, $a_{m,n} = 0$, when this case occurs. Once the number of the offloading task is equal to the current offloading capacity ($\pi_m$), the remaining tasks are assigned to being performed locally by the user's mobile terminal.

Consequently, in the worst case, the GBC-SFH iterates $N_m$ in the outer loop when there is no static offloading decision task. Which means a complexity of $N_m \cdot log(N_m)$ to sort the tasks, and $N_m \cdot K$ to assign to task at the inner loop. Thus, the maximum total number of iterations is $N_m^2 \cdot K + N_m^2 \cdot log(N_m)$. Therefore, the complexity of GBC-SFH is $\mathcal{O}(N_m^2 \cdot log(N_m))$, which is fast especially when the number of users associated to each AP is small [18, 19, 21] ($\leq 20$).

### 3.4.2 MEC computation offloading manager

The outer level of the Lagrangian dual problem is the master problem. It ensures a feasible offloading solution of the primal problem (3.18). Finding the optimal solution of the Lagrange dual requires an exhaustive search of all solutions' space and Lagrange multiplier values, which is difficult to do in general [66]. Consequently, we need

to adopt a faster approach. To deal with this problem, we propose the subgradient-based heuristic [65]. The proposed heuristic used in `the MEC computation offloading manager` has three steps, as illustrated in the algorithm 2. First, it solves the subproblems in the local offloading manager by the GBC-SFH for the current Lagrangian multipliers $\lambda$. Next, `the MEC computation offloading manager` checks if they obtained offloading solution is feasible. If not, the Lagrangian Adjustment Heuristic (LAH) will be used the get a feasible solution using local search. The idea of LAH heuristics is to check if every edge server respects the constraint $C5$. When a edge server does not respect this constraint, LAH heuristic reassigns some tasks offloaded from this edge server to other edge server that respects all constraints.

**Algorithm 2** *Pseudo code of MEC computation offloading manager*

**Input:**
$It_{max}$: maximum number of iterations ;
$\varepsilon$: an infinitesimal number;
**Output:** offloading decision and edge server selection for all users;
Initialize $\lambda^k$ randomly ;
$\mathcal{Z}_{max} = -\infty$ ;
**while** (t $< It_{max}$ and $\theta(t) > \varepsilon$) **do**
  $\mathcal{Z}_m(t)$ = get the solution of subproblem $m$ from the local offloading manager $m$, $\forall m \in \mathcal{M}$;
  $\mathcal{Z}(t) = \sum_{m \in \mathcal{M}} \mathcal{Z}(t)_m - \sum_{k \in \mathcal{K}} \lambda^k \cdot F^k$ ;
  **if** $(\mathcal{Z}(t) > \mathcal{Z}_{max})$ **then**
    $\mathcal{Z}_{feasible}$ = use heuristic $LAH$ to find a feasible solution ;
    **if** $(\mathcal{Z}_{feasible} < \mathcal{Z}^*)$ **then**
      $\mathcal{Z}^* = \mathcal{Z}_{feasible}$ ;
      update the best solution of the primal problem ;
    **end if**
    $\mathcal{Z}_{max} = \mathcal{Z}(t)$ ;
  **end if**
  update the Lagrange multipliers and $\eta$ using equations (3.25, 3.26 and 3.27);
**end while**

At the end, `the MEC computation offloading manager` updates the Lagrange multipliers by the following formula:

$$\lambda^k(t+1) = \lambda^k(t) + \theta(t) \cdot (\sum_{m}^{M}(\sum_{n}^{N_m} x_{m,n}^k \cdot c^k) - F^k) \tag{3.25}$$

where $\theta(t)$ is the update step. In this work, we use Held and Karp formula [65, 66] to

update this step as follows:

$$\theta(t) = \eta(t) \cdot \frac{\mathcal{Z}^* - \mathcal{Z}(t)}{\sum_{k=1}^{K}(\sum_{m}^{M}\sum_{n}^{N_m} x_{m,n}^{k} \cdot c^k - F^k)^2} \tag{3.26}$$

where $\eta(t)$ is a decreasing adaptation parameter $0 < \eta(0) \leq 2$, $\mathcal{Z}^*$ is the best obtained solution of the problem (3.18) and $\mathcal{Z}(t)$ refers to the current solution of the Lagrangian Dual. $\eta(t)$ can be expressed by the following formula:

$$\eta(t+1) = \begin{cases} \vartheta \cdot \eta(t) & \text{if } \mathcal{Z}(t) \text{ did not increase} \\ \\ \eta(t) & \text{otherwise} \end{cases} \tag{3.27}$$

As suggested in [65, 66] we set the values of $\vartheta = 0.9$ and $\eta(0) = 2$. The master problem repeats these steps until the stop conditions, which are the maximum number of iterations $It_{max}$ and the maximum tolerated error of the update step $\varepsilon$ ($\theta \leq \varepsilon$).

## 3.5 Numerical results analysis

In this section, we evaluate the performance of DM2-ECOP using the characteristics of realistic system configuration. We use a MEC environment consisting of a metropolitan area, which is composed of 20 APs forming a ring topology. The delay between any two APs is $3ms$ and the delay between every AP and the remote cloud is $100ms$ [18, 19]. We suppose that four edge servers are equidistantly deployed among the network, i.e: edge server 1 is collocated with the AP 1, edge servers 2 with the AP 6, edge server 3 with the AP 11 and edge server 4 with the AP 16. In order to study the performance of our offloading policy, we consider four edge servers configurations. Table 3.1 illustrates the list of the edge servers configuration considered for our tests. We consider real configurations, used by the public cloud provider such as: Amazon AWS and Microsoft Azure [21, 32, 33], to simulate the behavior of DM2-ECOP policy for real-world scenarios. The local computing capability of each user was randomly chosen from $f_{m,n} \in [0.8, 1, 1.2]$ Giga cycles/s.

Similar to [19], we assume that the number of users connected to every AP, $N_m$ is not greater than 20. Precisely, $N_m$ takes values from $\{5, 10, 15, 20\}$. Each user runs one application from the table 3.3, the first three applications are static offloading decision tasks, the others are dynamic offloading decision tasks [57]. The Table 3.2 details the

**Table 3.1:** *List of the edge servers' configurations used for the tests.*

| **Configuration** | **Computing capacity $F^k$ and allocation $c^k$ in Giga CPU cycles/s** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | edge server 1 | | edge server 2 | | edge server 3 | | edge server 4 | |
| | $c^1$ | $F^1$ | $c^2$ | $F^2$ | $c^3$ | $F^3$ | $c^4$ | $F^4$ |
| configuration 1 | 10 | 1000 | 10 | 1000 | 10 | 1000 | 10 | 1000 |
| configuration 2 | 15 | 1000 | 10 | 1000 | 15 | 1000 | 10 | 1000 |
| configuration 3 | 10 | 500 | 10 | 500 | 10 | 1500 | 10 | 1500 |
| configuration 4 | 15 | 500 | 10 | 500 | 15 | 1500 | 10 | 1500 |

wireless network parameter settings as presented in [37, 42, 43, 61]. For the backhaul network, we use the parameters presented in [68, 69]. Regarding the backhaul network, we use the parameters presented in [70].

**Table 3.2:** *The parameter setting of the network interface*

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $B_m$ | 50 Mhz | $b^{rx}_{m,n} = b^{tx}_{m,n}$ | $10^{-3}$ |
| $\mathcal{B}_m$ | 256 | $d_{m,n}$ | 10 |
| $\sigma^2_m$ | $5 \times 10^{-5}$ | $p^{tx}_{m,n}$ | 1.28 Watts |
| $p^{rx}_{m,n}$ | 1.18 Watts | $p^{idle}_{m,n}$ | 0.1 Watts |
| $p^{ap}_m$ | 10 Watts | $h^{tx}_{m,n} = h^{rx}_{m,n}$ | 0.1 |

The performances of DM2-ECOP are compared to two offloading policies from the literature:

- Nearest edge server Offloading (NCO) [18, 19]: in which each AP is associated to the nearest edge server. So, all the users connected to this AP offload their tasks to the same edge server. When a edge server is overloaded, the tasks are migrated to another edge server.
- Full Cloud Offloading (FCO) [31, 32]: in this case the users offload their tasks to the remote cloud. To make sense of the performances comparison of the offloading policy DM2-ECOP, NCO and FCO, we assume that the computing capacity allocated to perform each offloaded task in the remote cloud is 10 Giga cycles/s.

In the following, the default edge servers configuration is the first configuration (configuration 1) and the density of users at each AP is considered as the same, i.e., the same number of users at each AP. Furthermore, the stop criteria of the MEC computation offloading manager for DM2-ECOP are $It_{max} = 100$ for the maximal number of iterations, and $\varepsilon = 10^{-20}$ for the maximum tolerated error of update steps.

**Table 3.3:** *The characteristic of the real-world applications used for our tests.*

| **Application** | $\gamma_{m,n}$ (Giga CPU cycles) | $up_{m,n}$ (Kilobyte) | $dw_{m,n}$ (Byte) | $t_{m,n}$ (Second) |
|---|---|---|---|---|
| | static offloading decision tasks | | | |
| FACE | 12.3 | 62 | 60 | 5 |
| SPEECH | 15 | 243 | 50 | 5.1 |
| OBJECT | 44.6 | 73 | 50 | 13 |
| | dynamic offloading decision tasks | | | |
| Linpack | 50 | 10240 | 120 | 62.5 |
| CPUBENCH | 3.36 | 80 | 80 | 4.21 |
| PI BENCH | 130 | 10240 | 200 | 163 |

### 3.5.1 Convergence of DM2-ECOP

To evaluate the performance of DM2-ECOP and its convergence to a feasible solution, table 3.4 depicts the required number of iterations to get a feasible solution and the last value of update steps ($\theta$). As expected, the required number of iterations increase as the number of users in the network increases. Moreover, the update step converges to the maximum tolerated error $\varepsilon$ with a few number of iterations, this convergence changes while the number of users increases. Thanks to Held and Karp formula used in our work to update the Lagrange update step, for the rapid convergence of DM2-ECOP offloading policy.

**Table 3.4:** *The number of iterations and update step taken by DM2-ECOP to converge to a feasible solution.*

| number of users | number of iterations | update step $\theta$ |
|---|---|---|
| 100 | 15 | 0.0 |
| 200 | 20 | $9.12 \times 10^{-22}$ |
| 300 | 29 | $5.09 \times 10^{-21}$ |
| 400 | 43 | $4.46 \times 10^{-21}$ |

### 3.5.2 Offloading performance comparison

Figures 3.6 and 3.7 plot the offloading performances of DM2-ECOP, $NCO$ and $FCO$, if we set the cost parameter $\beta = 1$. We also distinguish between the costs related to the network access, network backhaul and processing. We note that DM2-ECOP reduces the total offloading cost compared to NCO and FCO. More precisely, we can see that the access cost of DM2-ECOP is the lowest, but the processing cost is the highest. This is due to the bandwidth allocation heuristic used by DM2-ECOP, which tries to

maximize the bandwidth allocated to each user by minimizing the offloading capacity ($\pi$) of each AP. So, less users can offload their tasks to the mobile edge computing with DM2-ECOP compared to *NCO* and *FCO*. However, where the wireless bandwidth is enough to offload all tasks, DM2-ECOP and *NCO* are equivalent, as shown in the figure where 100 users are in the network.

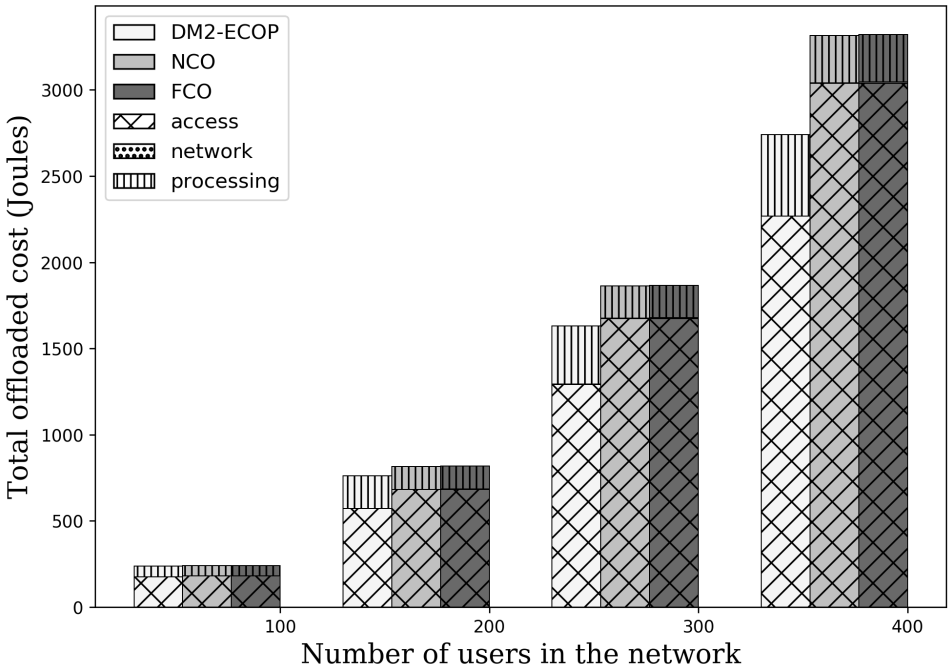



**Figure 3.6:** *Total offloading cost comparison of offloading policies DM2-ECOP, NCO and FCO where the cost parameter* $\beta_{m,n} = 1$.

To understand the effect of the users density on the offloading performances, we investigate in the Figure 3.8 the offloading gain of DM2-ECOP compared to NCO under different users density. We consider four scenarios where the topology is divided into two regions each one contains 10 APs. In scenario 1, the regions have the same user density. In scenario 2, users density in region 1 is twice the user density in region 2. In scenario 3, users density in region 1 is three times the user density in region 2. Finally, in scenario 4, users density in region 1 is four times the user density in region 2. In Figure3.8, we note that the offloading gain of DM2-ECOP compared to NCO goes up when to users density goes high. For example, where 200 users are in the network, the gain is 6.1% for scenario 1, and 10.5% for scenario 4. This is because the edge server selection in *NCO* is static, so this policy needs to migrate some tasks where

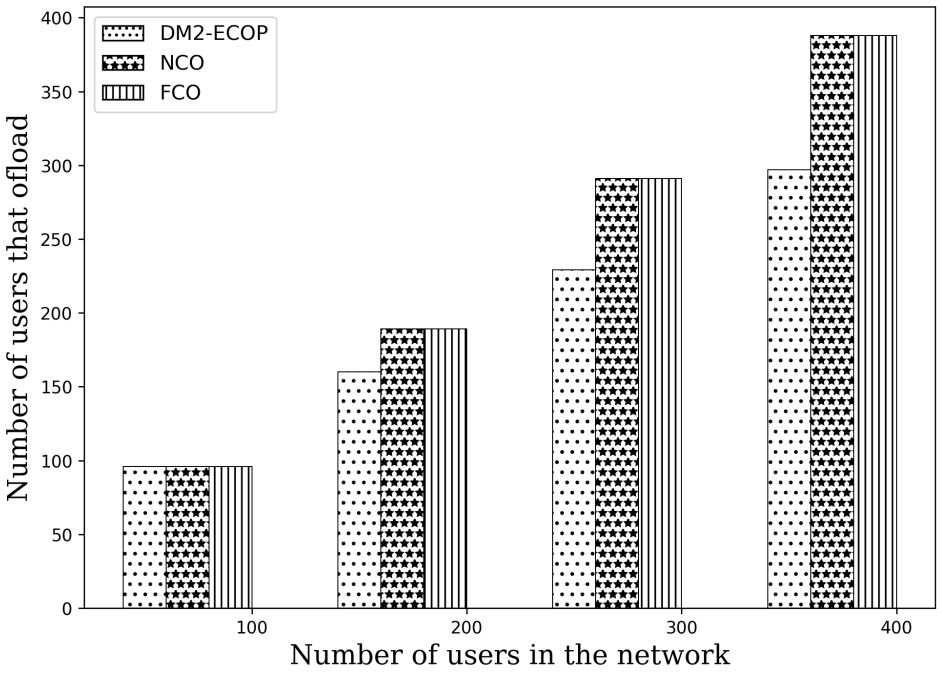



**Figure 3.7:** *The number of users that offload for the three policies DM2-ECOP, NCO and FCO where the cost parameter $\beta_{m,n} = 1$.*

the edge server is overloaded. Consequently, it adds an extra offloading cost. However, DM2-ECOP tries to find the best edge server selections dynamically at the offloading decision according to the system and network resource availability.

### 3.5.3 Impact of the cost parameter $\beta$ on the offloading performance

In Figure 3.9 studies the effect of the offloading cost parameter $\beta$ on the performance of the policies DM2-ECOP, $NCO$ and $FCO$. As we can see in the figure, the energy consumption of DM2-ECOP is the better than $NCO$ and $FCO$ for all possible values of $\beta$. Indeed, even if the we set $\beta$ to 0, which means that we give a complete priority to the tasks completion time, DM2-ECOP obtains better performances. Moreover, when we increase the value of $\beta$, the obtained performances are even better. Consequently, DM2-ECOP achieves better performance, in terms of energy consumption, whatever the offloading cost: energy consumption, completion time or a combination of energy and time. This is because the dynamic edge server selection adopted in DM2-ECOP.

In Figure 3.10 we investigate the effect of $\beta$ on the performance in terms of completion time.We note that the completion time of DM2-ECOP and NCO are better than FCO, this is because the edge servers are close to users. Moreover, The completion time of

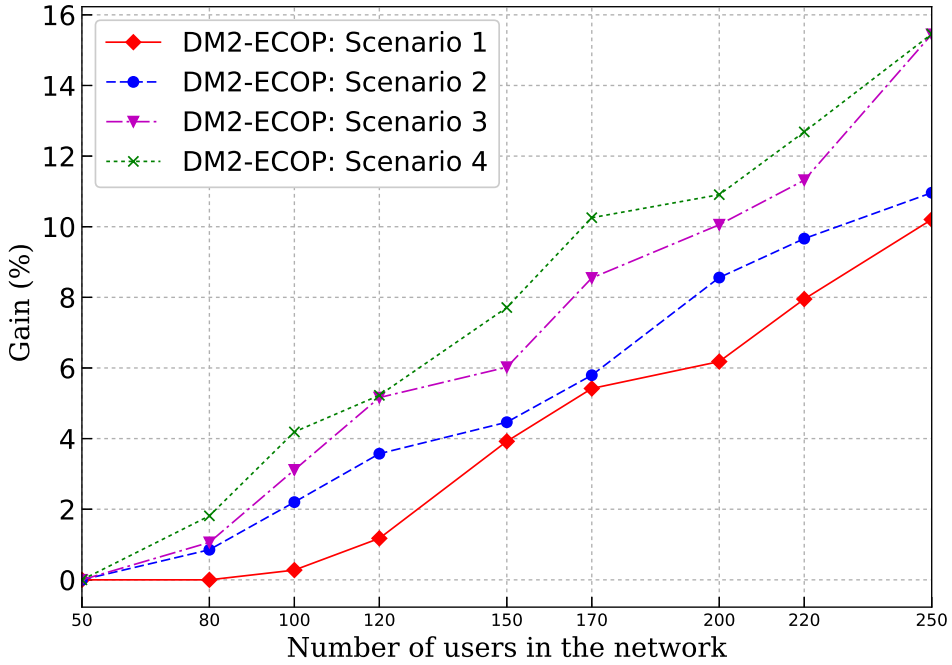



**Figure 3.8:** *Comparison of DM2-ECOP and NCO for different users density in the network*

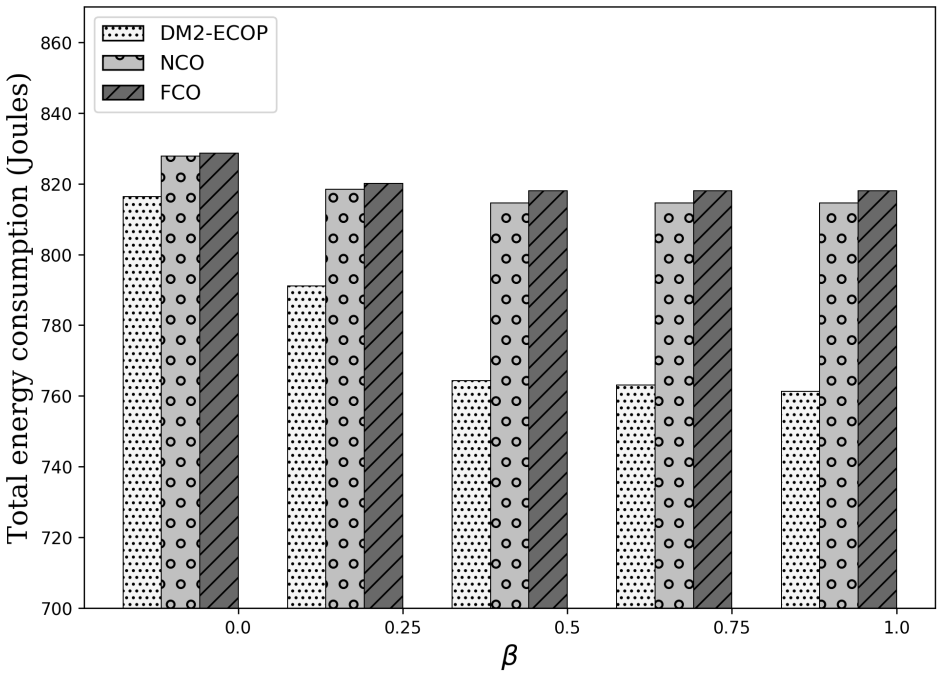



**Figure 3.9:** *Total energy consumption comparison for policies DM2-ECOP, NCO and FCO over the parameter $\beta$, where 200 users are in the mobile edge computing environment.*

DM2-ECOP is the lowest where $\beta = 0$. However, NCO achieves better completion time than DM2-ECOP where $\beta$ closes to 1. Consequently, NCO has the best performance in terms of completion time. In fact, when $\beta$ is close to 1, the energy consumption becomes more important than the completion time in the expression of the offloading cost. DM2-ECOP reduces the offloading cost by offloading less tasks to the MEC, as shown in 3.7, in order to minimize the energy consummated by the wireless access level. As a result, more tasks are executed locally, which increase the completion time.

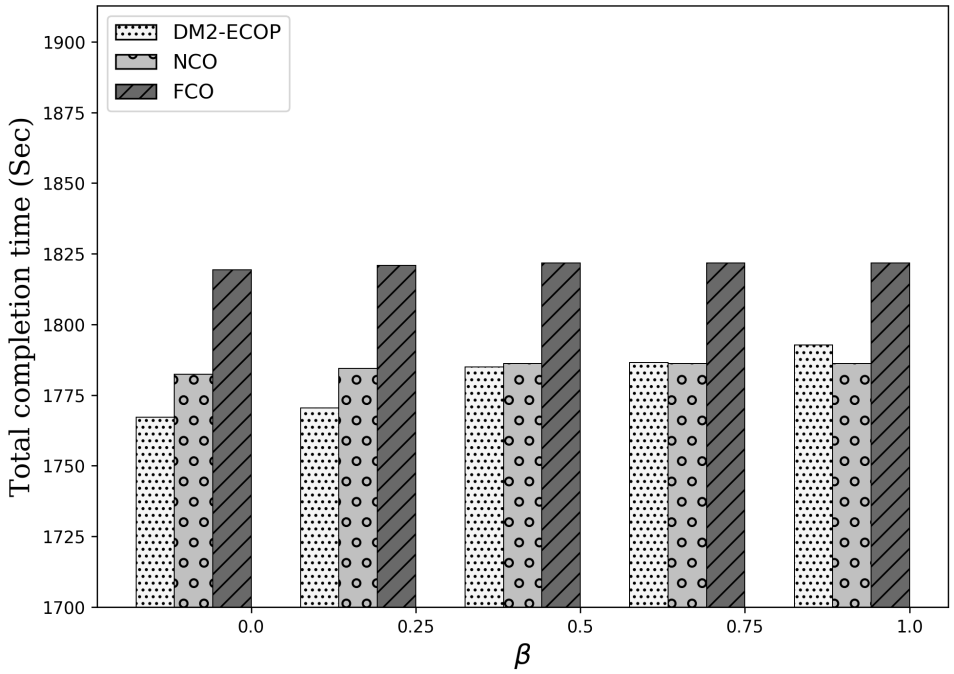



**Figure 3.10:** *Total completion time comparison for policies DM2-ECOP, NCO and FCO over the parameter $\beta$, where 200 users are in the mobile edge computing environment.*

### 3.5.4 Impact of the edge servers configuration

In the following, we study the performance of the offloading policies over heterogeneous edge servers configurations. In Figures 3.11 and 3.12, we investigate the performance of the offloading policies DM2-ECOP and *NCO* over four edge servers configurations presented at the beginning or this section. We observe that the DM2-ECOP and *NCO* are equivalent when the edge servers have exactly the same configurations, such as configuration 1. However, where the computing resources allocated to each task are heterogeneous, which corresponds to more realistic scenario, DM2-ECOP achieves better performance in terms of energy consumption and completion time. This can

be explained by the fact that DM2-ECOP offloads to the best edge server, but *NCO* offloads to the nearest edge server. We also can notice that DM2-ECOP gets better performance for configuration 2 than configuration 4, even if the resource allocated for each task are the same in the two configurations. This is because of the total computing capacity in configuration 4 is not homogeneous. Thus, NCO needs to migrate some tasks from edge servers 1 and 2 to edge servers 3 and 4. To summarize, these results show that DM2-ECOP can achieve a good offloading performance under different edge server configuration.

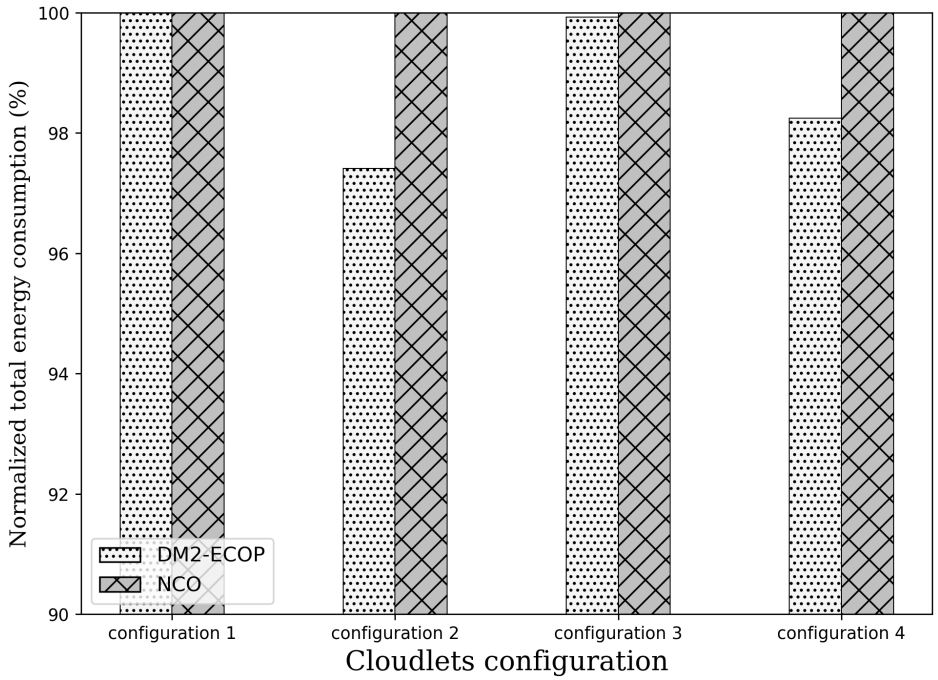



**Figure 3.11:** *Total energy consumption of policies over different edge servers configurations, where $\beta = 1$ and 200 users are in the mobile edge computing environment.*

### 3.5.5 Impact of the applications characteristics

As shown previously in our analytical model, the characteristics of the application have a crucial role in the efficiency of the offloading performance. To understand this role, we investigate the impact of the application on the offloading cost and on the amount of the offloaded computation, $a_{m,n}$.

Figure 3.13 and 3.14 depict the performances of the offloading policies for each application under the edge servers configuration 3. We note that for static offloading decision tasks, DM2-ECOP has better energy consumption and completion time followed

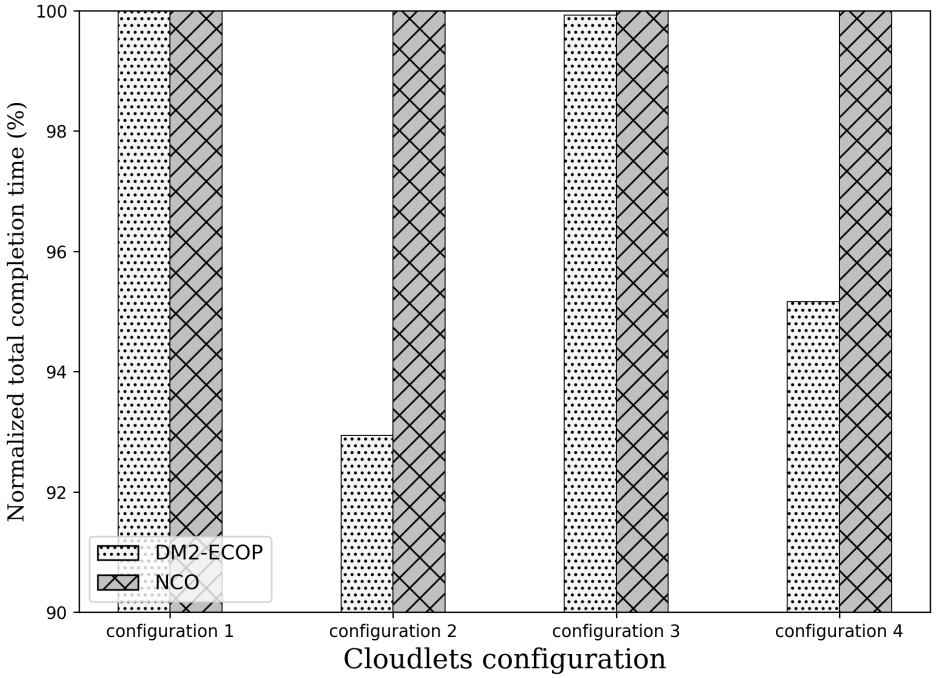



**Figure 3.12:** *Total completion time of policies over different edge servers configurations, where $\beta = 1$ and 200 users are in the mobile edge computing environment.*

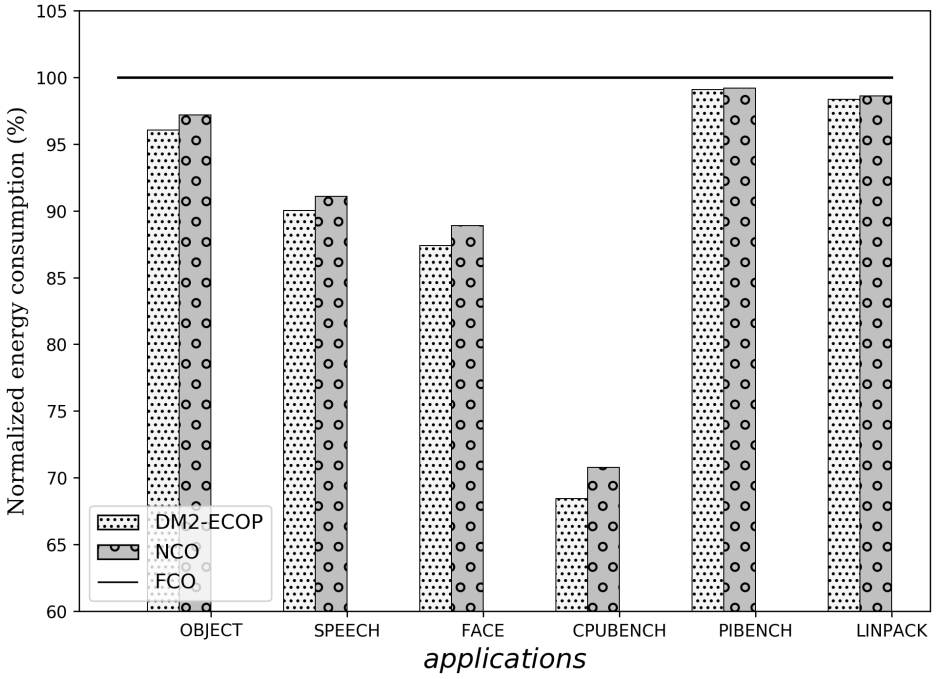



**Figure 3.13:** *Total energy consumption of policies for each application for the configuration 3, where $\beta = 1$ and 200 users are in MEC.*

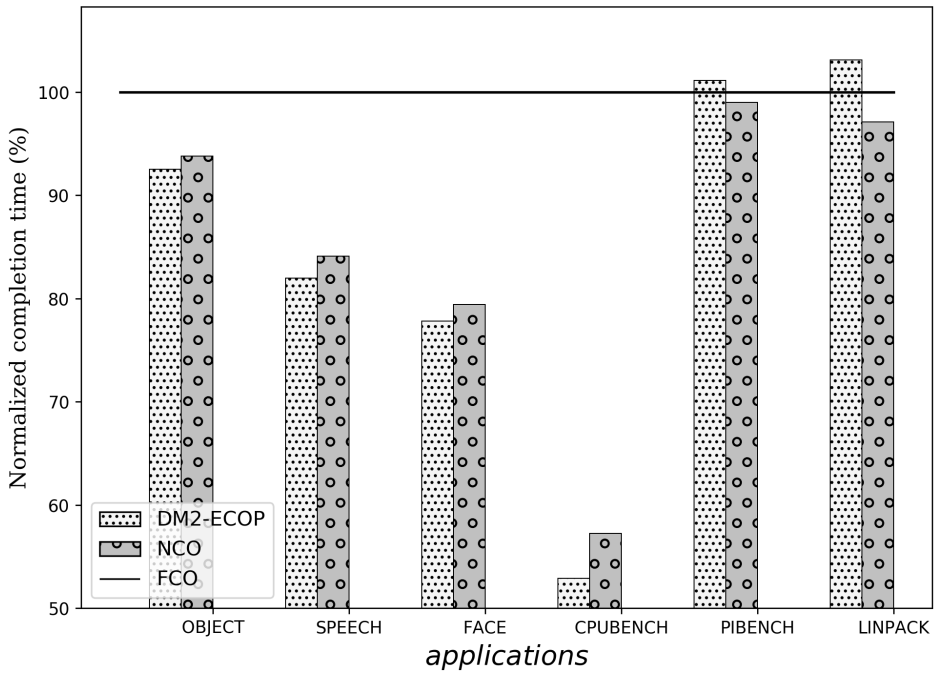



**Figure 3.14:** *total completion time of policies for each application for the configuration 3, where $\beta = 1$ and 200 users are in MEC.*

by $NCO$. Indeed, the static offloading decision tasks must be offloaded, DM2-ECOP tries to find the best edge server selection at the decision time. However, FCO offloads to the nearest edge server, which induces an additional offloading cost since some tasks must be migrated to other edge servers. On the other hand, for the dynamic offloading decision tasks, we note that DM2-ECOP has the lowest energy consumption and completion time where the application does not need to upload lots of data to the edge server, such as CPUBENCH. However, where the application requires lots of data, such as LINPACK and PIBENCH, the completion time of $NCO$ and $FCO$ are better than that of DM2-ECOP. This is because of DM2-ECOP tries to perform the application locally, when it uploads a lot of data in order to minimize the access cost of the users.

Finally, to analyze the performance of DM2-ECOP with the dynamic offloading decision tasks, we study the effect of the number of users and the ratio between the remote and the local processing capacity on the offloading decision. In Figure 3.15, we plot the effect of the number of users per AP and thus the amount of bandwidth allocated to each user on the offloading decision. The red area corresponds to the case where we execute a task only locally (i.e. $a_{m,n} = 0$) and the green area corresponds to

the case where a task is totally offloaded (i.e. $a_{m,n} = 1$). In addition to the offloading decision, we also plot the upload data-computing ratio $\psi_{m,n}$ for three dynamic offloading decision tasks, namely Linpack, CPUBENCH and PI BENCH. As we can see, the three applications do not behave the same way when we increase the number of users per AP. Indeed, Linpack is the most sensitive application and stop offloading when the number of users is greater than 14. On the other hand, CPUBENCH is the less sensitive application since it stops offloading when the number of users reaches 37. This is due to the fact that Linpack sends much more data when it offload compared to CPUBENCH.

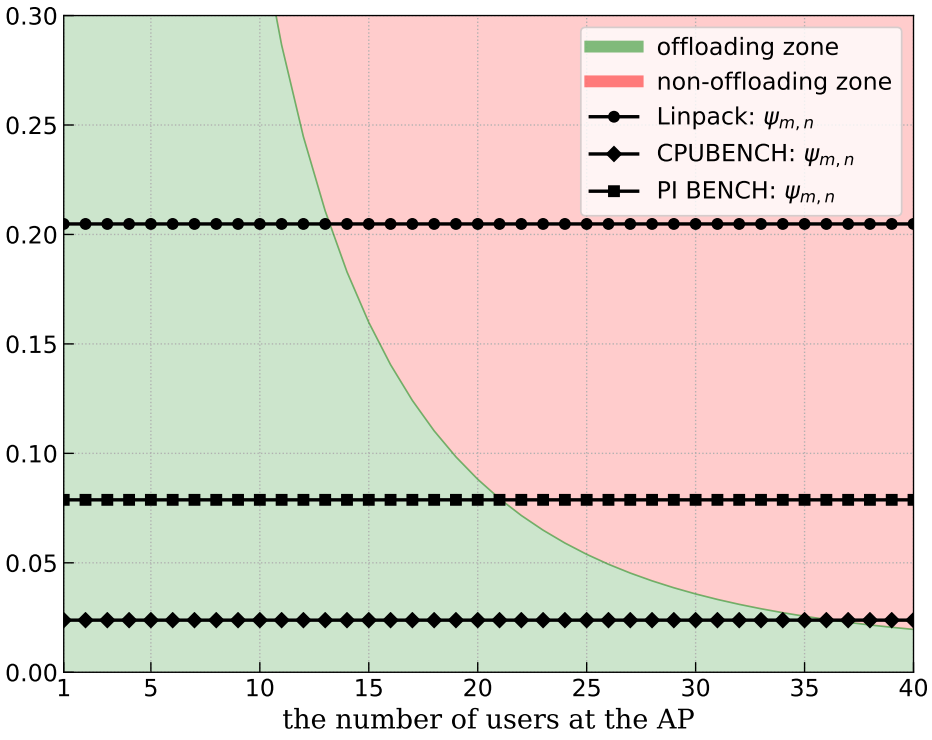



**Figure 3.15:** *The effect of the number of user connected to AP on the application partition decision, where the local cpu power of user terminal is $f_{m,n} = 1$ Giga Cycles.*

In Figure 3.16, we investigate the impact of the ratio between the local and the remote processing capacity on the offloading decision. As in the Figure 3.15, we also plot the upload data-computing ratio $\psi_{m,n}$ for Linpack, CPUBENCH and PI BENCH. As we can see, more the remote processing capacity is important compared to the local processing capacity more the decision is to offload task. However, as in last figure, Linpack is less sensitive to that increase compared to CPUBENCH and PI BENCH. Indeed, since the total amount of data that should be offloaded for Linpack is

important, the offloading becomes beneficial only if the remote processing capability is very important in comparison of the local processing capability.

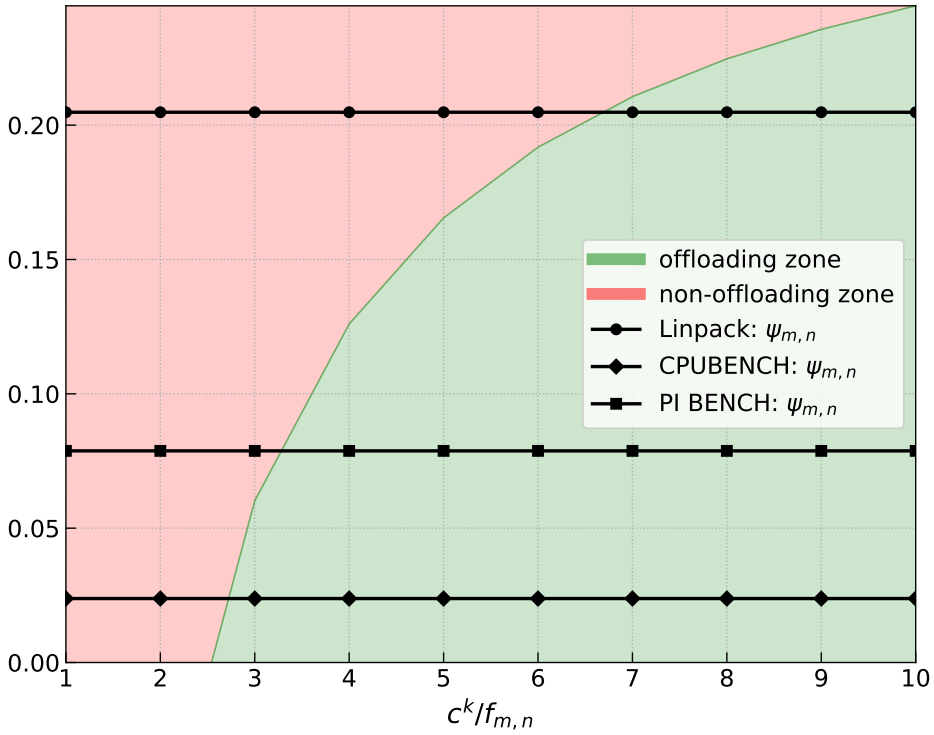



**Figure 3.16:** *The effect of the allocation computing resource $c^k$ on the application partition decision, where the local cpu power of user terminal is $f_{m,n} = 1$ Giga Cycles.*

## 3.6 Conclusion

In this chapter, we explored the computation offloading in multi-user multi-edge server MEC environment. Our main objective was to understand the effect of the edge server selection and application characteristics on the performance of the offloading. A new computation offloading approach was designed to enhance the existing approaches. Our proposed approach, DM2-ECOP, implements an offloading policy which decides the users that should offload and selects an edge server to perform the tasks of each user that offload. In addition, DM2-ECOP takes into consideration two categories of applications, applications that must be performed remotely in MEC and applications that can be performed both locally and remotely. For the latter category of applications, DM2-ECOP must compute the amount of computation that should be offloaded to MEC. The obtained numerical results show performance improvements in terms of the

offloading cost compared to existing offloading policies under different scenarios and edge servers configurations. Moreover, because we consider that all the tasks have the same priority and that they are not sharing resources (same CPU) at the edge server, we demonstrate that the best choice is to offload all the computation to the MEC or perform all of it locally.

Even though that DM2-ECOP increases the performance of the application, it considers only the edge servers available in the local network. Thus, in a real-world MEC where the edge servers communicate with the remote cloud, DM2-ECOP does not capitalize on the remote cloud unlimited resources. The MEC's infrastructure may be misused in such a scenario. In the next chapter, we extends our approach to include the remote cloud in the offloading decision.

CHAPTER 4

# Extension of D2M-ECOP to Two-tier MEC

*ECESO offloading approach*

**Abstract**

This chapter investigates the computation offloading over two tiered MEC environment. It extends the work presented in chapter 3 by adding a second tier to the MEC, which is the remote cloud. The objective is to minimize the total energy consumed by the users and understand the effect of the remote cloud on the computation offloading in MEC. Moreover, it presents an experimental proof of the importance of our new offloading approach. The numerical and experimental results show that our proposal achieves better performance than existing solutions [71, 72].

[71] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud". In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2018, Montreal, QC, Canada, October 28 - November 02, 2018*. 2018, pp. 137–145. DOI: 10.1145/3242102.3242133

[72] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud". In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2018, Montreal, QC, Canada, October 28 - November 02, 2018*. 2018, pp. 137–145. DOI: 10.1145/3242102.3242133

## 4.1 Introduction

In the chapter 3, we highlight the edge server selection and computation offloading decision in multi-user multi-edge server MEC environment. The proposed offloading policy, D2M-ECOP, improves the performance in terms of both energy consumption and completion time of the application. However, it is designed for a MEC environment composed only of edge servers, which is not always a realistic scenario. In fact, a real-word MEC environment is built on two-tier architecture [73, 74], where the edge servers provide limited computing capabilities with ultra-low latency to the end user while the remote cloud offers infinite computing resources with a considerable internet latency [71, 73, 74].

The difference in the computational capabilities configuration makes the offloading decision more challenging in such environment. Indeed, the decision whether a user offloads its computations to an edge server or to the remote cloud can highly impact the performance of the MEC [71]. Exactly, the placement of the users' computation on the MEC environment has a crucial role in the efficiency of the offloading performance in two-tier MEC environment [71, 73, 74]. To deal with this issue, we introduce, in this chapter, a new version of D2M-ECOP offloading policy, that we named Efficient edge server (Cloudlet) Selection Offloading policy, ECESO. ECESO extends D2M-ECOP to consider two-tier MEC environment. Basically, ECESO assigns each user to the best server, edge server or the remote cloud, in the MEC in order the reduce the energy consumption of all users, according to the network and system resources availability.

Besides that D2M-ECOP approach achieves good performances in numerical results, it will be important to consolidate our conclusion via real-world experiments. Thus, in this chapter, we explore the computation offloading for real-systems settings in order to evaluate our proposal. We designed and implemented a new computation offloading middleware for Android-based terminals for the purpose to have experiments with real Android applications. This middleware was integrated on both the mobile terminal and the offloading server. We integrated our offloading policy to the mobile terminal via a client offloading middleware in order to decide if the task should be executed locally or offloaded to the edge/remote cloud. In the remote virtual machine, we also integrated a server offloading middleware to ensure that the offloaded tasks are correctly executed. Using this testbed, we were able to run multi-users offloading experiments in local edge and in the cloud, with the aim to compare the observed performances to the placement

decision made by our proposals.

The chapter is structured as the following. Section 4.2 presents the MEC environment infrastructure and offloading system configurations. Section 4.3 illustrates the offloading for policy two-tier MEC. The performance and evaluation of ECESO is depicted, analyzed and discussed in section 4.4. Section 4.5 details the computation offloading middleware developed in order to evaluate ECESO with real-world experiments, then it analyzes and discusses the results of the experiments. The last section 4.7 draws a conclusion on the importance of ECESO offloading approach in the two-tier MEC environment.

## 4.2 Two-tier MEC environment description and modeling

As in the chapter 3, we consider the MEC environment illustrated in Figure 4.1. The MEC's infrastructure is composed of $M$ APs, $K$ deployed edge servers and one remote cloud. In the remainder, we will refer to the remote cloud as the $(K+1)^{th}$ server. As in the previous chapter, let $\mathcal{M}$ denote the set of APs. The remote cloud and the edge servers constitute a set, denoted $\mathcal{K}$, of servers. All these servers offer computation resources to perform offloaded tasks. In each server $k \in \mathcal{K}$, the resources are characterized by a fixed capacity, denoted $F^k$, of computational resource units. A computational resource unit is expressed in Ghz and is defined as the number of cycles per second allocated to perform a task. We denote by $c^k$ the number of cycles per second allocated by server $k$ to perform any given offloaded task. Similarly to [32, 33], we consider that $c^k$, $\forall k \in \mathcal{K}$, is fixed and does not change during the computation. We also assume that the number of computation resources units, $F^k$, is limited in edge servers while it is infinite in the remote cloud. Formally, $\forall k \in \{1,2,...,K\}$, $F^k \ll F^{K+1} = \infty$.

For the rest of the MEC characteristics, we use the same as in chapter 3. In addition, due to hardware and software constraints required by the task, we assume that some edge servers cannot perform some tasks. In this case, we define another binary variable, $g^k_{m,n}$, to indicate if the edge server $k$ can perform the task $(m,n)$. Thus $g^k_{m,n}$ is equal to 1 when the $k^{th}$ edge server can perform the task of user $(m,n)$, 0 otherwise. Without lose of generality, we assume that the remote cloud can perform all the tasks whatever the type of the task. So, $g^{K+1}_{m,n} = 1, \forall (m,n)$.

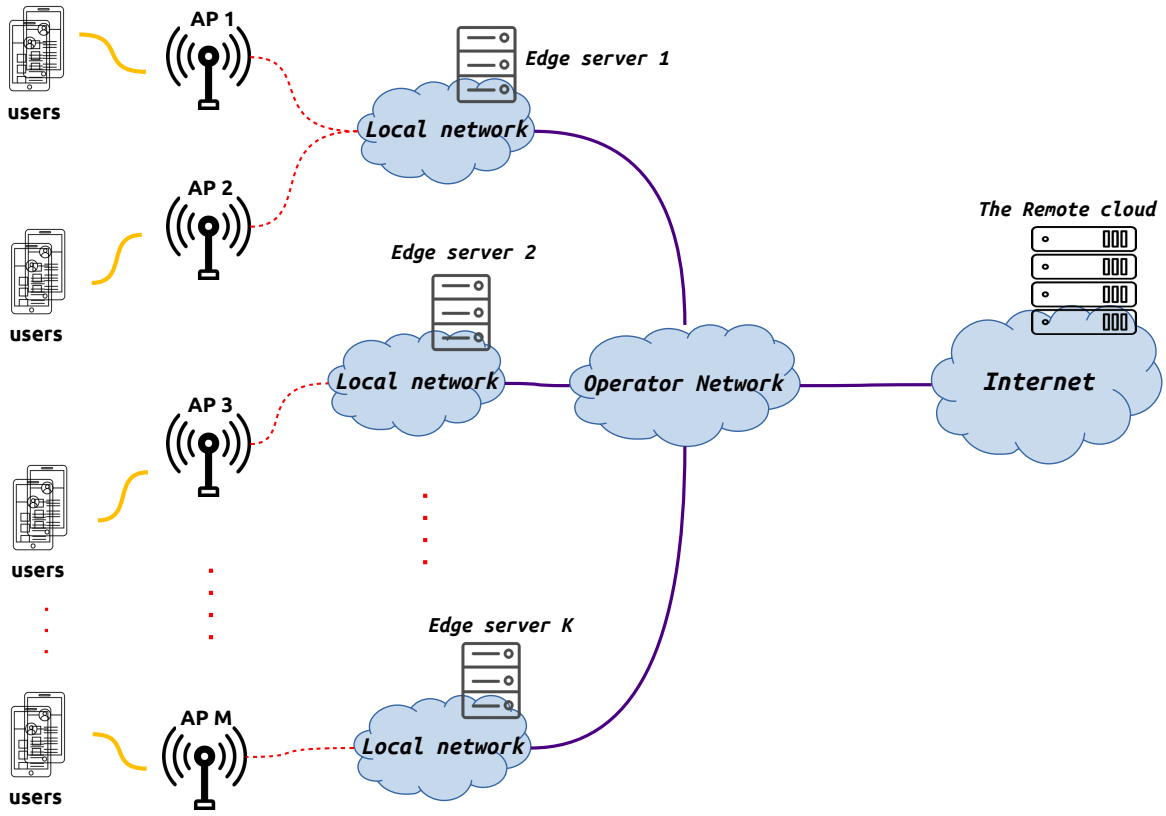



**Figure 4.1:** *An overview of the two-tier MEC environment.*

## 4.3 Problem formulation and solution

To solve the computation offloading problem, we extend our offloading approach proposed in the chapter 3. The new version, maned ECESO, takes on consideration the available edge servers around uses, and the unlimited computing resource of the remote cloud. It tries to offload more tasks in order to reduce the total energy consumption of the users' terminals. Basically, ECESO uses the offloading policy presented in section 3.4 of chapter 3 with some modifications for the remote cloud. In the local offloading manager, ECESO uses the heuristic GBC-SFH, illustrated in Algorithm 1, by considering only the edge server. At this stage the edge servers have more importance than the remote cloud. In MEC Computation Offloading manager algorithm, ECESO uses the heuristic LAH presented in Algorithm 3. This heuristic insures that the obtained offloading solution is a feasible one that satisfies the constraint $C5$ of the optimization problem 3.18. It starts by checking that all the edge servers are satisfying the constraint. If not, LAH reassigns the overloaded tasks in the edge server to the best server, edge server or the remote cloud, which gives the minimum energy consumption of the task. In a second step, LAH checks if the offloaded to the remote cloud is better for each task. If so, the

task will be reassigned to the remote cloud. The remaining of the algorithm is the same as DM2-ECOP.

**Algorithm 3** *Pseudo code of LAH heuristic for ECESO approach*

**Input:**

The solutions of subproblem $m, \forall m \in \mathcal{M}$;

**Output:** The cost of a feasible offloading solution $\mathcal{Z}$;

$F^k_{allocated} = \sum_{m \in \mathcal{M}}^{n \in \mathcal{N}_m} (x^k_{m,n} \cdot c^k), \forall k \in \mathcal{K}$;
$stop = False$;
**while** $(!stop)$ **do**
  **if** (for each edge server $k$ $F^k_{allocated} < F^k$) **then**
    $stop = True$;
  **else**
    Select the edge server $k$ that has $F^k_{allocated} < F^k$;
    Compute the priority $\xi_{m,n}$ of each task $(m,n)$ offloaded to edge server $k$ using equation 3.23;
    Sort the tasks in increasing order of $\xi_{m,n}$;
    **while** $(F^k_{allocated} > F^k)$ **do**
      $(m,n) \longrightarrow$ the task with the lowest $\xi_{m,n}$;
      Reassign the task $(m,n)$ to the to the server $k^{'} \in \mathcal{K} - k$ that satisfies the constraint $C5$ of problem 3.18;
      Update $F^{k^{'}}_{allocated}$ and $F^k_{allocated}$ ;
    **end while**
  **end if**
**end while**
If a task $(m,n)$ is offloaded to MEC, reassign it to the remote cloud if the energy consumption will be improved and constraint $C3$ defined in 3.17 is respected;

## 4.4 Numerical results and analysis

This section focuses on the evaluation of the performance of ECESO offloading policy by evaluating several performance metrics, e.g. the average number of offloaded tasks, and the energy saving under different settings. The MEC environment consists of a metropolitan area is composed of 20 APs, four edge servers already deployed in the network and one remote cloud. In addition, two network typologies are considered. The ring topology, in which the edge servers are equidistantly deployed in the network, i.e: edge server 1 is collocated with the AP 1, edge servers 2 with the AP 6, edge server 3 with the AP 11 and edge server 4 with the AP 16. The second topology is generated by GT-ITM [75] tool, where the edge servers are randomly deployed. In order to get a better understanding of the offloading policies performances, we consider real mobile applications shown in Table 3.3 of chapter 3. In addition, we consider two edge servers configurations. In the first configuration, each edge server has a computing capacity

of 1000 Giga cycles/s, and allocates 10 Giga cycles to perform every offloaded task ($F^k = 1000$ and $c^k = 10$). In the second configuration, we consider heterogeneous edge servers, where edge servers 1 and 2 have a computing capacity of 500 Giga cycles/s and edge servers 3 and 4 have a computing capacity of 1500 Giga cycles/s. Similar to the previews chapter, we use the network configurations illustrated in Table 3.2. Moreover, as in [19], we assume a homogeneous users distribution in the network.

In order to evaluate the performance of ECESO approach, we compare its results with existing approaches that consider two-tier MEC in computation offloading. They can be detailed as follows:

- **DOTA** [19, 21]: In **DOTA**, each AP is associated with the nearest edge server. In this case, all users connected to this AP offload their tasks to the same edge server. When a edge server is overloaded, the tasks are migrated to the remote cloud.
- **CBL** [18, 26]: Using **CBL** we also associate each AP to the nearest edge server. Thus, all users connected to that AP have to offload their tasks into that same edge server. However, unlike **DOTA**, when the edge server is overloaded, the tasks are migrated to another edge server.
- **FCO** [31, 32]: In this policy, all users offload their tasks to the remote cloud.

In order to compare these offloading policies, we also define comparison metrics depicting the gain of a given offloading policy $\mathcal{P}$. This gain represents the benefit of the offloading policy $\mathcal{P}$ compared to case where the task is offloaded to the cloud (i.e. **FCO** policy). We formulate the gain as following:

$$\text{gain of } \mathcal{P} = 100 * \frac{(\text{cost of FCO} - \text{cost of } \mathcal{P})}{\text{cost of FCO}}$$

### 4.4.1 Effect of edge server configuration

In Figure 4.2, we plot the gain of ECESO compared to the gain of DOTA and CBL, when considering a network topology following the configuration 1 with homogeneous edge servers characteristics. As expected, we can observe that the gain decreases when the number of users increases. This is manly due to the fact that the backhaul cost increases when the number of offloaded tasks. We can also observe that the performance of ECESO, DOTA and CBL are almost equivalent, except when the number of users

exceeds 300, where we notice that our approach is slightly better. This is due to the fact that ECESO tries to maximize the bandwidth allocated to each user, and it offloads in priority the tasks with the greatest impact on the cost. Consequently, less tasks are offloading compared to the other offloading policies.

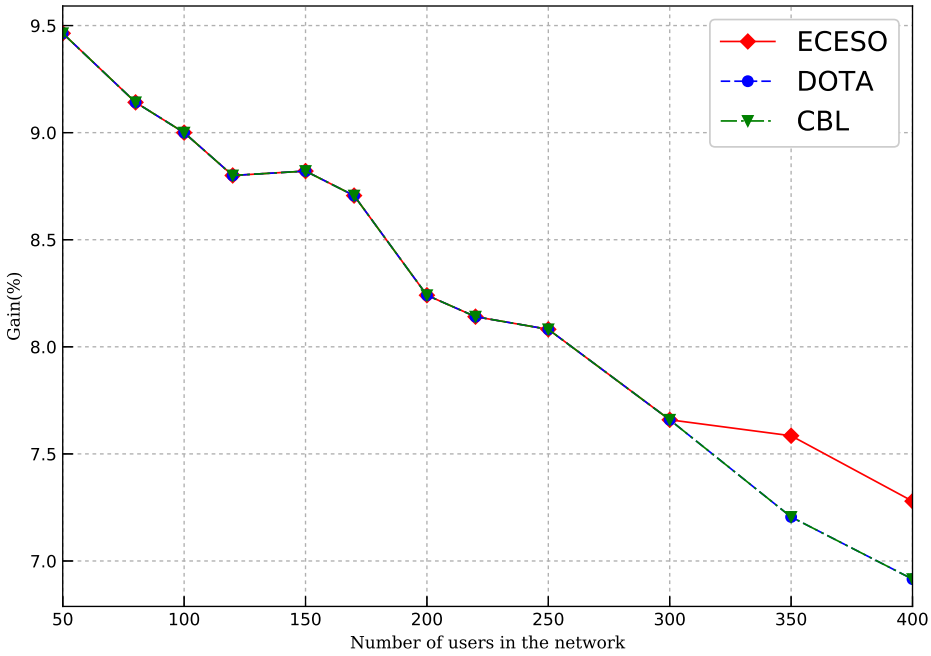



**Figure 4.2:** *Offloading gain over homogeneous edge servers configuration.*

In Figure 4.3, we compare the performances of ECESO, BCL and DOTA in the case where heterogeneous edge servers are deployed for the ring topology. As we can see, when few number of users are considered ($< 100$), the performances of the three policies are equivalent. However, when the number of users increases ECESO outperforms BCL and DOTA, for both of network topologies' configurations. This is due to the fact that, when we increase the number of users both edge servers 1 and 2 cannot support all the offloaded tasks. In this case, DOTA policy start to migrate the overloaded tasks to the remote cloud, which adds more offloading costs. On the other hand, CBL tries to migrate the overloaded tasks from edge servers 1 and 2 to edge servers 3 and 4, and also add some additional offloading cost but less than DOTA. However, using ECESO, for each task, we can select the best edge server at the offloading decisions step, which reduce the additional offloading cost due to the migration of tasks introduced in DOTA and CBL. Finally, we notice that the network topology affects the performance

of ECESO more than DOTA and CBL. For example, the gain for ECESO is 8.5, 8.03% for ring topology and GT-ITM topology consequently. However, it is 7.5, 7.34% for CBL, and 5.92, 5.81% for DOTA. This is due to the fact that ECESO uses the topology to select the edge servers. Nevertheless, DOTA and CBL assign statically the AP to the edge server.

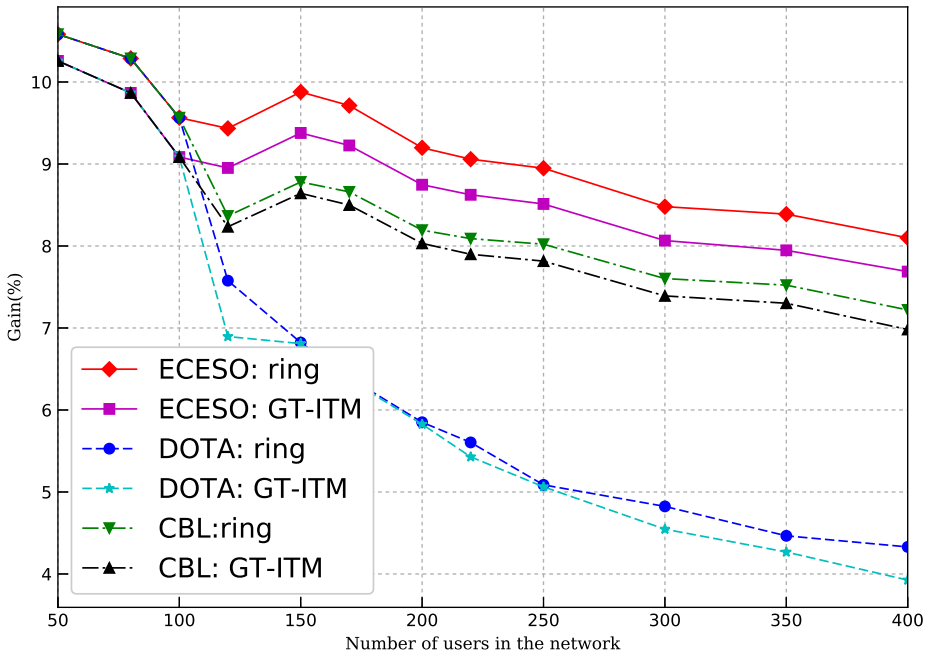



**Figure 4.3:** *Offloading gain over heterogeneous edge servers and network topology.*

### 4.4.2 Edge servers Vs remote Cloud computing resources

Now, we investigate how each application behaves according to the amount of resources allocated to perform it in the edge servers and in the cloud. Precisely, we fix the computing capacity allocated to each user in the edge servers to 10 Giga cycles/s and we vary this capacity between 10 to 20 Giga cycles/s in the cloud. In Figure 4.4, we illustrate the behavior of static offloading decision tasks FACE, SPEECH, and OBJECT. As we can see, all of them are always offloaded to the edge servers or the cloud have the same computation computing 10 Giga cycles/s. However, where the remote cloud has greater computing capacity than edge servers ($c^{cloud} > c^{edgeservers}$) all the tasks are offloaded to the remote cloud.

Figures 4.5, 4.6 and 4.7 illustrate the performances of the ECESO policy for *dynamic*

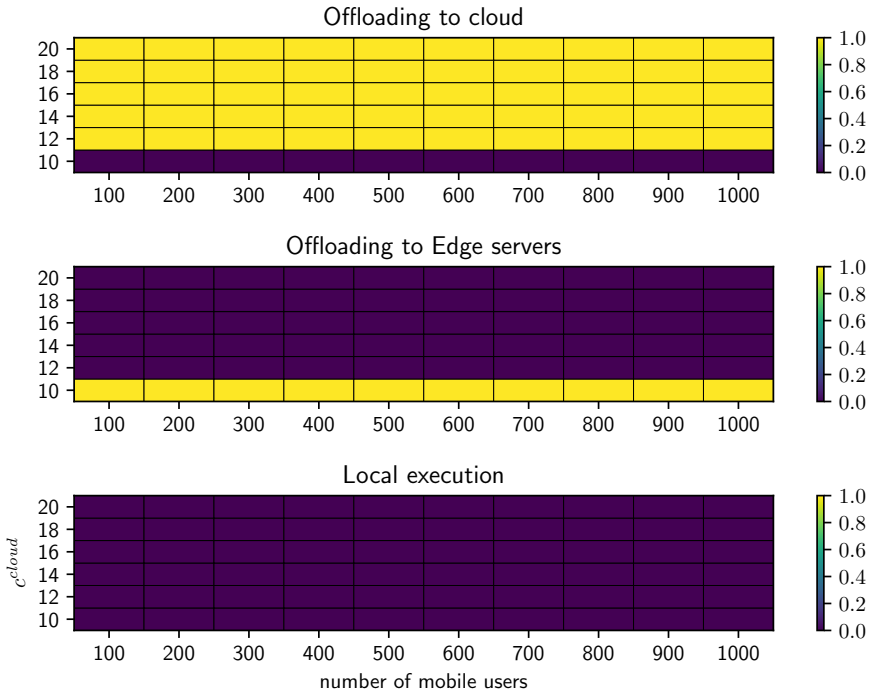



**Figure 4.4:** *The location of the execution of tasks under different users and cloud computing capacity ($c^{cloud}$ in Giga cycles/s), case of static offloading decision apps.*

*offloading decision tasks*, CPUBENCH, PIBENCH and LINPACK, respectively. The ratio of the offloading tasks decreases when the number of the users in the network increases, for example, for CPUBENCH application 100% of tasks are offloaded where the number of users is not greater than 200, but only 30% are offloaded where 1000 users are in the network. This decreasing of the ratio of offloaded tasks is due to the wireless bandwidth in the AP. We also note that the ratio of the offloaded tasks depends on the application characteristics when the applications require a lot of computing resources and transmit a huge amount of data (Linpack and PIBENCH) the ratio of offloaded tasks is lower.

As a conclusion, offloading tasks on the edge server or the remote cloud depends on many factors, in the figures we noticed that the computing resource allocation in the remote cloud, the computing resource allocation in the edge servers and the characteristics of the application affect directly the placement of the offloaded tasks.

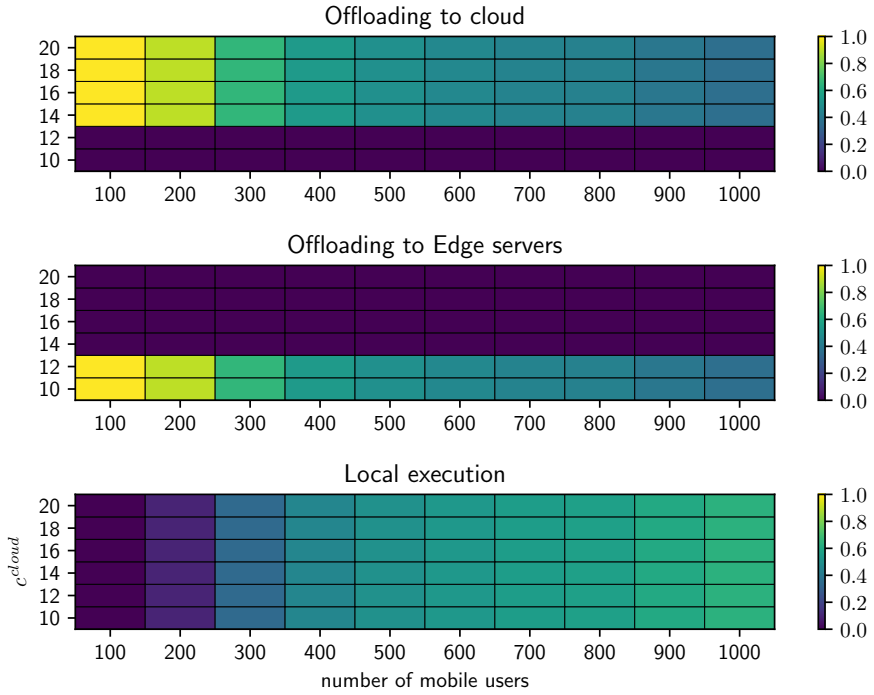



**Figure 4.5:** *The location of the execution of tasks under different users and cloud computing capacity ($c^{cloud}$ in Giga cycles/s), case of CPUBENCH.*

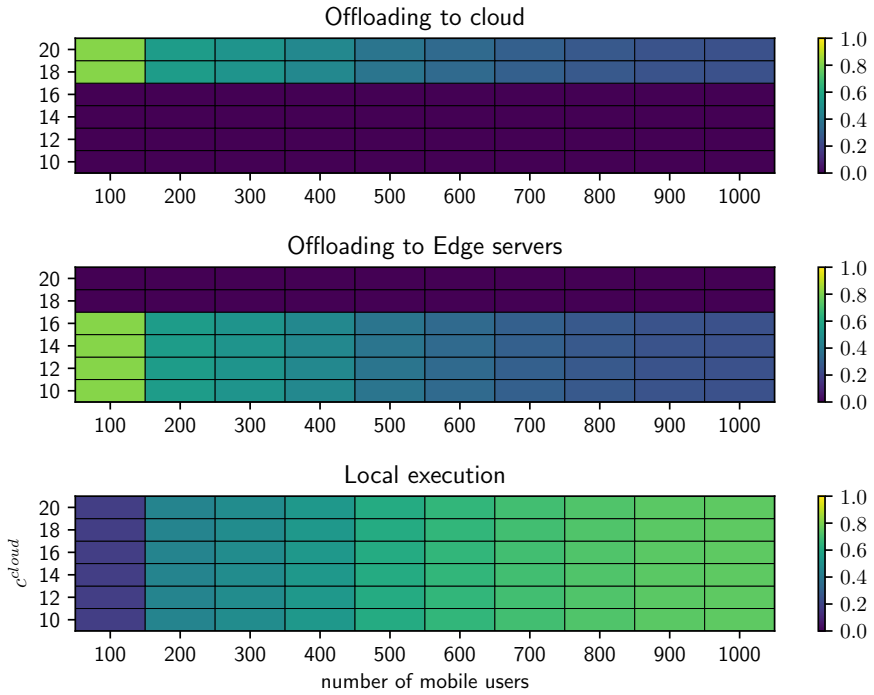



**Figure 4.6:** *The location of the execution of tasks under different users and cloud computing capacity ($c^{cloud}$ in Giga cycles/s), case of PIBENCH.*

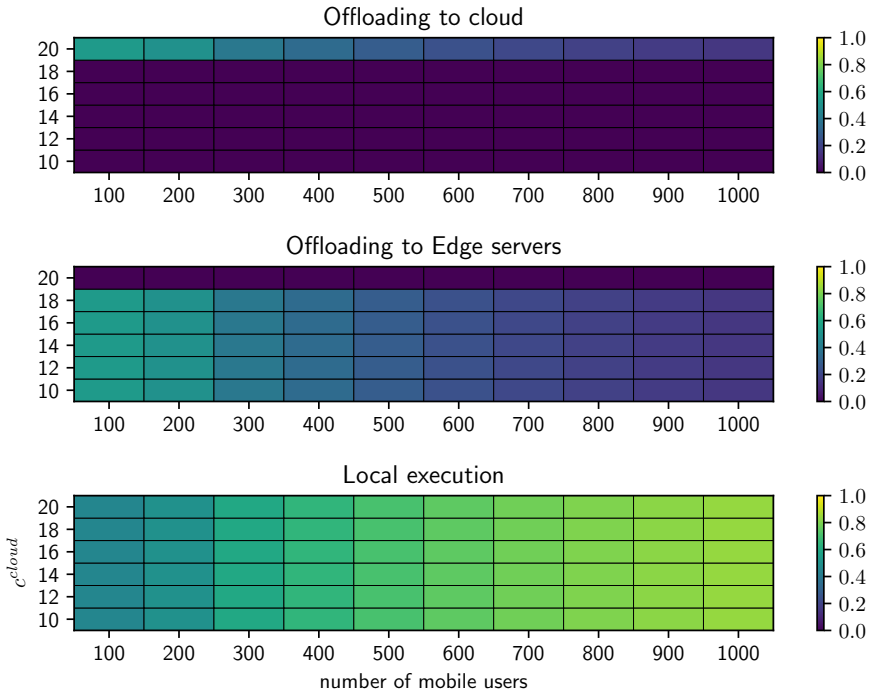



**Figure 4.7:** *The location of the execution of tasks under different users and cloud computing capacity ($c^{cloud}$ in Giga cycles/s), case of LINPACK.*

## 4.5 Experimental results and analysis

In addition to the numerical evaluation, we also need to evaluate the behavior of our offloading policy using real mobile applications. In the following, we describe the tested environment that we consider as well as our proposed offloading middleware.

### 4.5.1 Platform description

As illustrated in Figure 4.8, our experiments platform regroups several hardware and software components, as described in the following.

#### 4.5.1.1 Mobile terminal

Even if our testbed can be used with real Android smartphones, we decided to consider an Raspberry Pi 3 device. The main idea is to have a controlled experimental environment, since we need to measure the energy consumed by the device and also to control the bandwidth for the mobile terminal. We used a Raspberry Pi 3 Model B (RPi3) powered by a Quad Core Broadcom BCM2837 64bit ARMv8 processor at 1.2 GHz and 1GB LPDDR2 of RAM at 900 MHz. On this device, we installed Android 8 "Orio" as operating system and also our middleware, described in section 4.5.1.3. Finally, we developed and implemented three dynamic offloading decision tasks applications: Linpack,

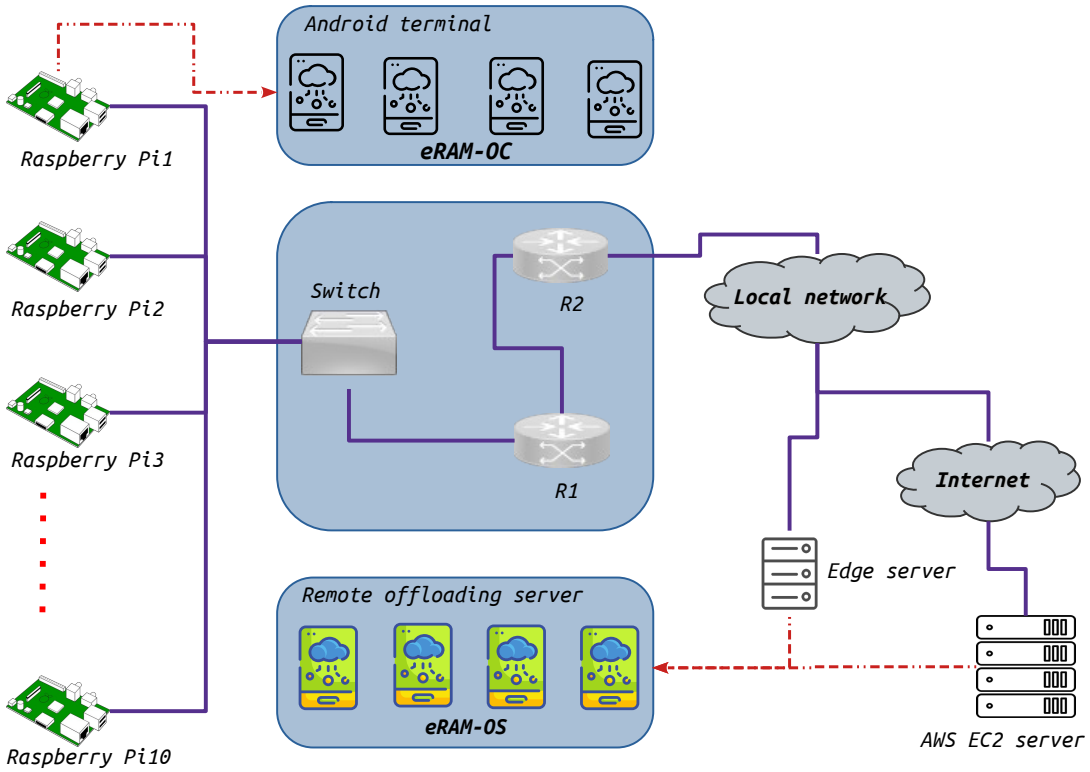



**Figure 4.8:** *An Overview of the testbed environment.*

CPUBENCH and PI BENCH. Each of these bench applications was implemented as a fragment within a common Java Android application. The size of the `APK` source file is 5427.2 Kilobytes.

#### 4.5.1.2 Offloading server

The offloading server can be an edge server or an EC2 instance in AWS cloud provider. We used a desktop PC powered by an `Intel I7-6700 8 cores CPU` at `3.40 GHz` and `16GB DDR3` of RAM with an `Ubuntu 18.04.2 LTS` as operating system as the edge server. In order to run native Android applications, we also installed, using VirtualBox, a virtual machine with Android-x86 distribution as operating system. Android-x86 [76] is an open source project to port Android on x86 platform. For the remote cloud, we consider AWS Amazon Cloud using the EC2 `t2.medium` instance. This instance is powered by high-frequency Intel Xeon processors with 2 vCPU and a memory of 4 GB. In both AWS EC2 instance and the the local virtual machine, we install the server eRAM-OS of offloading middleware.

#### 4.5.1.3 Computation offloading middleware

We develop a new computation offloading middleware named eRAM, Elastic Remote task Acceleration for resource-constrained Mobile terminals, for Android-based applications using java programming language. This middleware allows the execution of some part or the totality of an application by a remote server in the MEC environment. It transmits the source code and the input data of the offloaded tasks from the mobile terminal to the target MEC edge server. As illustrated in Figure 4.9, eRAM has two main components each is responsible of a single role of the middleware. The client component, which is running by the Android terminal to ensure the offloading of the applications. And the server component, which is running by the edge servers in order to ensure the execution of the offloaded tasks and applications. Both of the client and server communicate with each other via the network in order to exchange the information and data. Inspired by the offloading middleware from literature [46, 53, 55, 77] to make eRAM extensible and easy to manage, it is implemented following the component-based design pattern [78]. Each of the components is composed of many modules, every module has a single function and the module communicate between themselves for the working of the middleware.

eRAM-OC, or eRAM Offloading Client, is the middleware component one the mobile terminal side. It automates the offloading policy of the application. As detailed above, eRAM-OC is composed of four main modules that can be detailed as the following:

- **`Profiler manager:`** The profiler collects the mobile terminal information that will be used to improve the offloading decision. It is composed of three modules, each one gathers a type of data. The system profiler collects the data of the terminal such as: CPU usage, CPU frequency and the battery levels. The application profiler gathers the data of the application such as: the processing time, the source code size, the input data size and the number of instruction in the application. Lastly, the data of the network such as: the bandwidth, the latency, the upload size and downloaded data size are collected by the network profiler.

- **`Offloading manager:`** This is the core module of the middleware, it implements the algorithms of the offloading policy. Firstly, it estimates the offloading cost using the equations 3.13 and 3.16 and based on the information gathered by the profiler. It determines, then, the offloading decisions of the applications to decide

if the offloading is benefits or not. Finally, it schedules the application's tasks based on the offloading decisions. The offloaded tasks are transmitted to the target edge server and the rest are performed locally.

- **`Communication manager:`** The communication manager takes in charge the communication between the two modules of the eRAM platform. It uses advanced network sockets to communicate between the client and the server. Also, it uses zeroconf [79] protocol to discover the nearby edge servers around the mobile terminal.

- **`Logger:`** The logger receives the gathered information and saves them in the database. In addition, it can select the data from the data base in order to uses them by the Offloading manager.

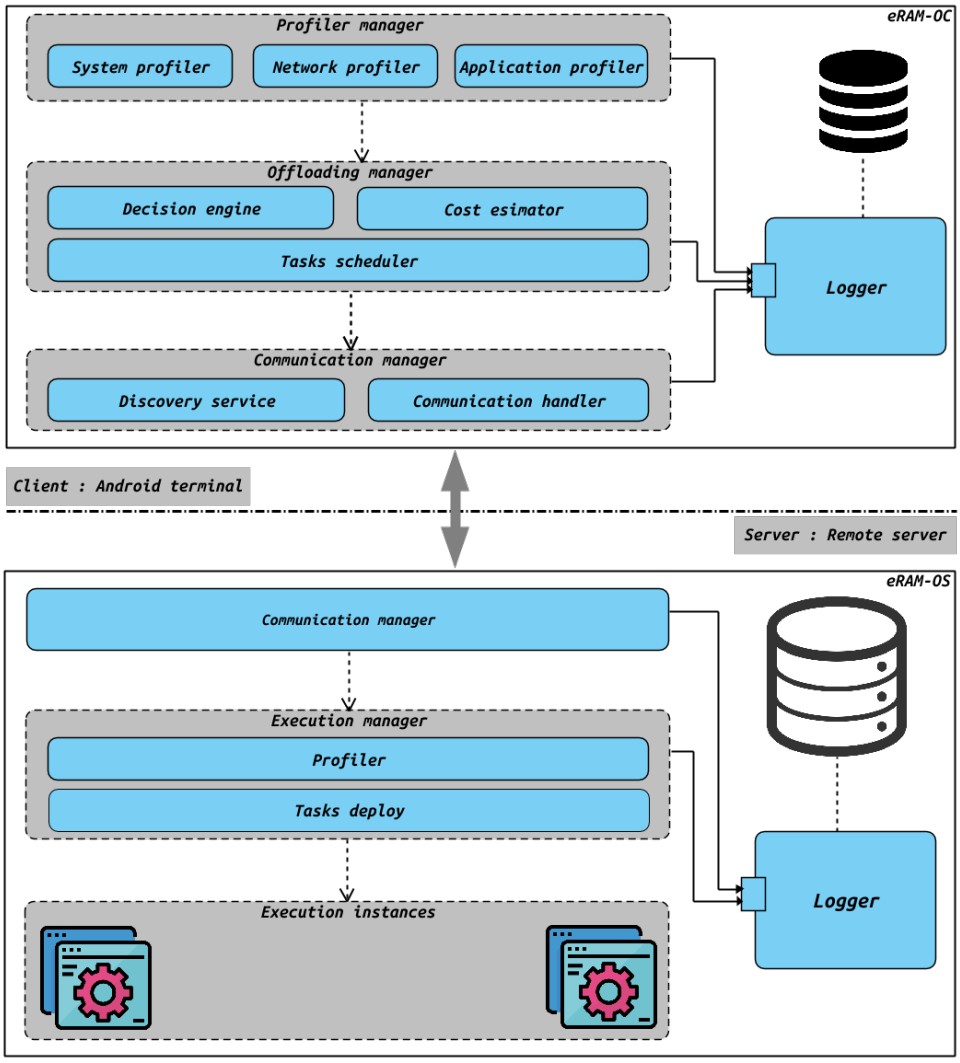



**Figure 4.9:** *An overview architecture of eRAM computation offloading middleware.*

Th server module eRAM-OS, for eRAM Offloading Server, is the part of the middleware running in the remote MEC server. It organizes the processing of the offloaded tasks, and returns the results to the mobile terminal. As in the Figure, eRAM-OS is composed of three illustrated as follows:

- **`Communication manager:`** This component ensures the communication with the eRAM-OC of the mobile terminals that offload to the edge server. It receives the offloaded tasks, send them to been performed in the edge server, and returns the results to the mobile terminal.

- **`Execution manager:`** The execution manager handles the processing of the offloaded tasks in the edge server. It deploys the received tasks on the executing instances. In addition, it monitors the execution's information, such as: execution time and allocated instance, using the profiler.

- **`Logger:`** Similarly to the logger at eRAM-OC, this module receives the gathered information and save them in a database. These information can be used by the Execution manager in order to improve the processing of the tasks in the edge server.

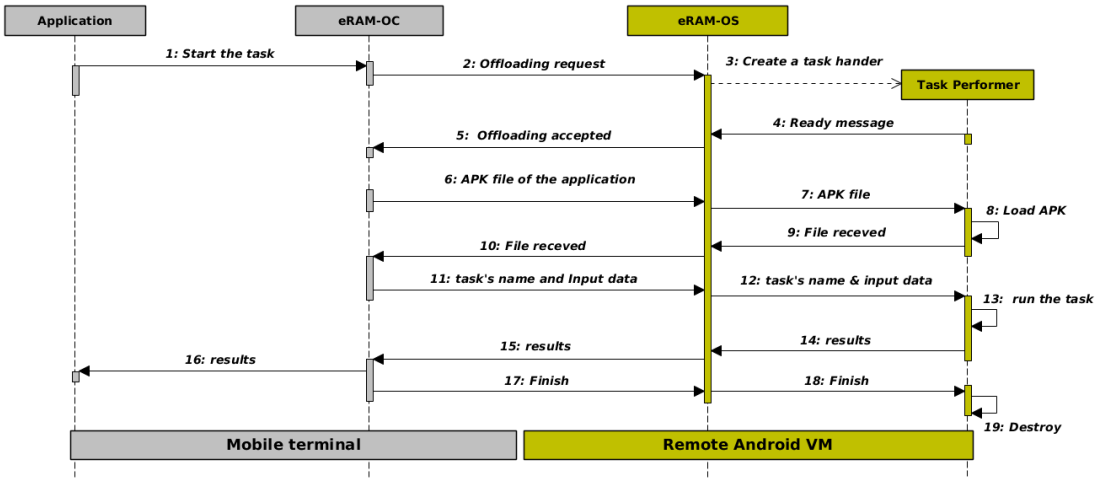



**Figure 4.10:** *The protocol computation offloading of eRAM middleware.*

In figure 4.10, we illustrate the offloading steps. As we can see, when the user needs to offload an application a set of messages are exchanged between the eRAM-OC and eRAM-OS. In the following, we summarize the main steps and the objective of each message.

- **`Request an offloading authorization:`** As we can see in messages (steps 2, 3, 4 and 5), the eRAM-OC sends an offloading request message to eRAM-OS to request an offloading authorization. The eRAM-OS checks the resources availability and if so it creates a new task handler. Finally, it sends back an acknowledgment message to the eRAM-OC to indicates that the acceptance of the offloading request.

- **`Source code uploading:`** The second set of messages (steps 6, 7, 8, 9 and 10) are responsible for uploading and installing the APK source code of the application from the eRAM-OC to the eRAM-OS.

- **`Entry point and input data uploading:`** After the source code, the user transmit (steps 11 and 12) task's execution entry point in addition to the required input data.

- **`Results:`** after the processing of the task, the eRAM-OS sends back (steps 14, 15 and 16) to the eRAM-OC the task execution results.

- **`End of the offloading:`** The last step is the transmission of an end message (steps 17 and 18) in order to inform the eRAM-OS of the end of the offloading in order to destroy the Task handler.

#### 4.5.1.4 Mobile applications characteristics

Our first experiments was to characterize the benchmark of the three applications: Linpack, CPU-Bench and PI-Bench. Each of these applications was parameterized in order to study the system under three CPU resource requirement situations: 1) *Light*, 2) *Medium* and 3) *Full*. Precisely, Linpack is a software that performs matrix calculations. To generate *light* processing requirements, we parameterized Linpack with 17 square matrix sizes, ranging from 1 to 17. *Medium* and *full* configurations were generated by multiplying the above matrix sizes by a factor of 40 and 70, respectively. CPU-Bench is a software that generates a random floating point and a random integer of a predefined size $n$. To generate light, medium and full configurations, we set $n = 10^6$, $10^7$ and $10^8$, respectively. Finally, three CPU resource requirement situations for Pi-Bench are obtained by computing *pi* with an approximation of $10^3$, $10^4$ and $10^5$ decimal places. It is important notice that, our results are average values obtained after executing 10 runs with the same experiments setting.

Using our testbed, we measured the number of CPU cycles ($\gamma_{m,n}$), the quantity of uploaded data ($up_{m,n}$) and the quantity of downloaded data ($dw_{m,n}$) of the three bench applications, under *Light*, *Medium* and *Full* configurations. The results are shown in Table 4.1. One can observe that all the measured parameters are different from those reported from the literature [57] in table 3.3, except downloaded data for Linpack. New versions and the possibility to execute these applications by setting some parameters might explain such a gap. Notice also that due to our implementation of these three bench applications as fragments within the same Java Android application, the quantity of the downloaded data is always equal to the size of the `APK` source file (5427.2 bytes). Finally, we observe that among the three bench applications, Linpack is the one that generates the lowest number of CPU cycles. Under *Full* configuration, it is also the one that leads to the highest number of CPU cycles.

**Table 4.1:** *The characteristic of the bench apps*

| **Application** | $\gamma_{m,n}$ (Giga CPU cycles) | $up_{m,n}$ (Kilobyte) | $dw_{m,n}$ (Byte) |
|---|---|---|---|
| Linpack | | | |
| Light | 0.2033 | 5427,2 | 120 |
| Medium | 547.406 | 5427,2 | 120 |
| Full | 2909.11 | 5427,2 | 120 |
| PIBENCH | | | |
| Light | 2.203 | 5427,2 | 56 |
| Medium | 45.539 | 5427,2 | 56 |
| Full | 1310.882 | 5427,2 | 56 |
| CPUBENCH | | | |
| Light | 8.696 | 5427,2 | 30 |
| Medium | 293.613 | 5427,2 | 30 |
| Full | 745.435 | 5427,2 | 30 |

#### 4.5.1.5 Network topology

To connect the Android terminals to the remote offloading servers, we have set up a network topology composed of two Cisco routers. The first router is connected to the client via a switch (LAN) and the second router allows our testbed to be connected to the edge server through a local network and to the remote cloud via an Internet connection. To connect the two routers, we use a serial link allowing us to control the speed at which the data, in bits per second (bps), is sent between the two routers. The main objective is to control the bandwidth between the Android terminals and the

remote offloading server. In our experiments, we used the bandwidth values offered by the serial link which are: 1.2$Kbps$, 4.8$Kbps$, 9.6$Kbps$, 38.4$Kbps$, 72$Kbps$, 125$Kbps$, 500$Kbps$, 5.3$Mbps$ and 8$Mbps$.

### 4.5.2 Single-user scenario

Figure 4.11 shows the energy consumption measured in our testbed for Linpack *easy*, when it is executed locally on the mobile terminal, on the edge with 1 or 2 vCPU and on the clouds located at Paris or at the Ohio, both allocating 2 vCPU. The $x$ axis represents the bandwidth of the serial link that we have varied from 1.2 Kbps to 8 Mbps. Notice that logarithmic scale is used for $x$ and $y$ axis. As one can see, when the offered bandwidth is restricted to less than 500$Kbps$, the lowest energy consumption is obtained for local execution. Above 500$kbps$, offloading Linpack *easy* on the edge with 2vCPU offers better energy performances. Starting from 5.3$Mbps$ offloading on the cloud located at Paris outperforms the local execution. These experimental results prove that, even for applications that require light processing resources, offloading could be energetically beneficial when the available bandwidth to the remote server is sufficiently high.

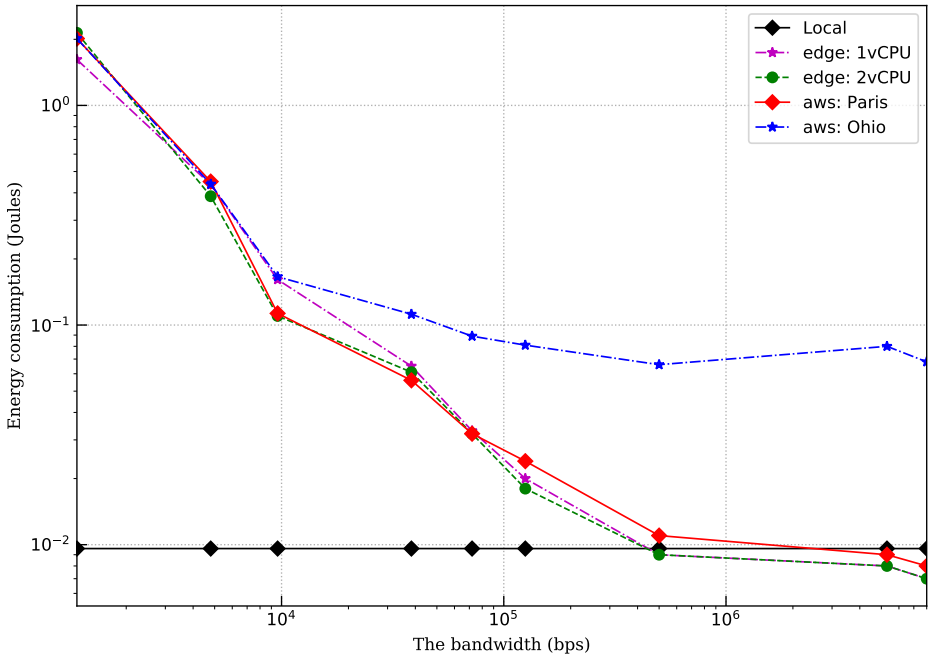



**Figure 4.11:** *Linpack easy: Energy Consumption*

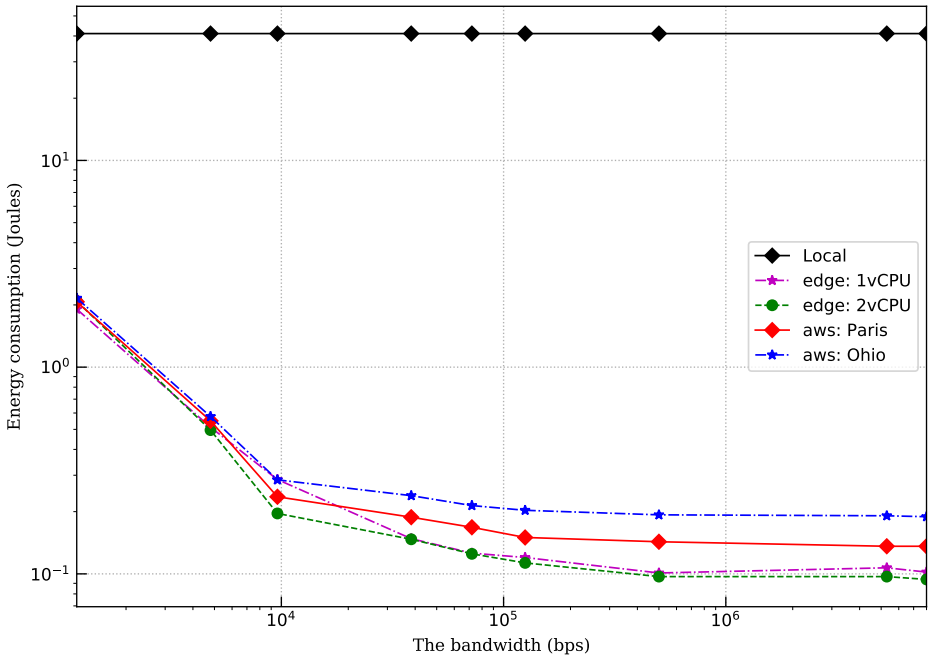



**Figure 4.12:** *Linpack medium: Energy Consumption*

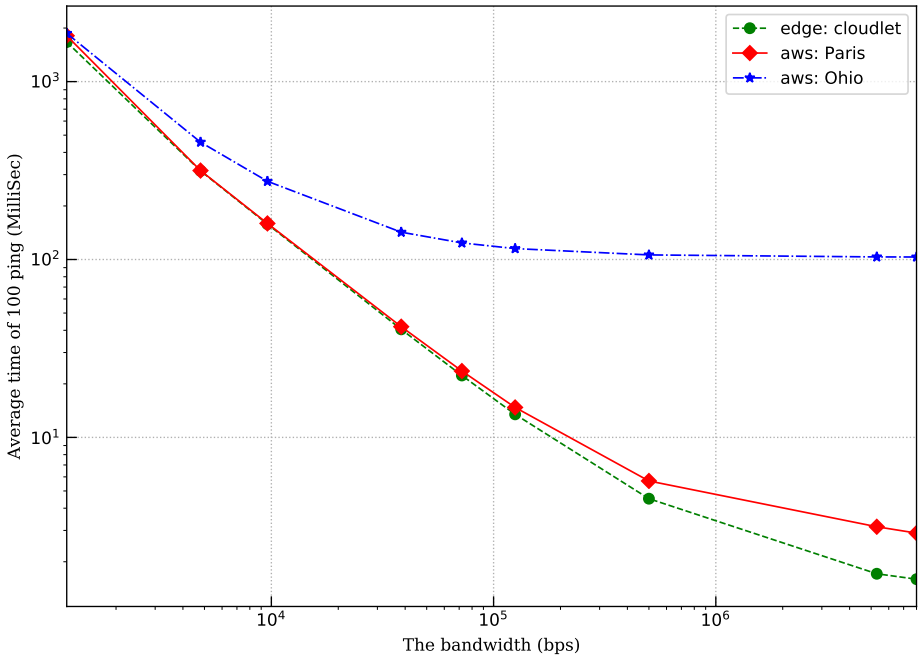



**Figure 4.13:** *Round Trip Time between the terminal and different locations*

Another situation fostering the remote execution is observed when the task's processing needs is important. This is clearly illustrated for Linpack *medium* in figure 4.12. We can notice a certain hierarchy in the performances: first the closest edge, then the remote cloud, and lastly the local execution. A noticeable discrepancy can be observed in figure 4.12 between the clouds at Paris and at Ohio. We believe that this is related to differentiated network latency from our lab to these two clouds. The average Round Trip Time (RTT) measured from the terminal toward different remote servers is drawn in figure 4.13. We can see that the RTT for the edge and the cloud in Paris are very close to each other, while the RTT to the cloud in Ohio increases significantly for larger bandwidth values.

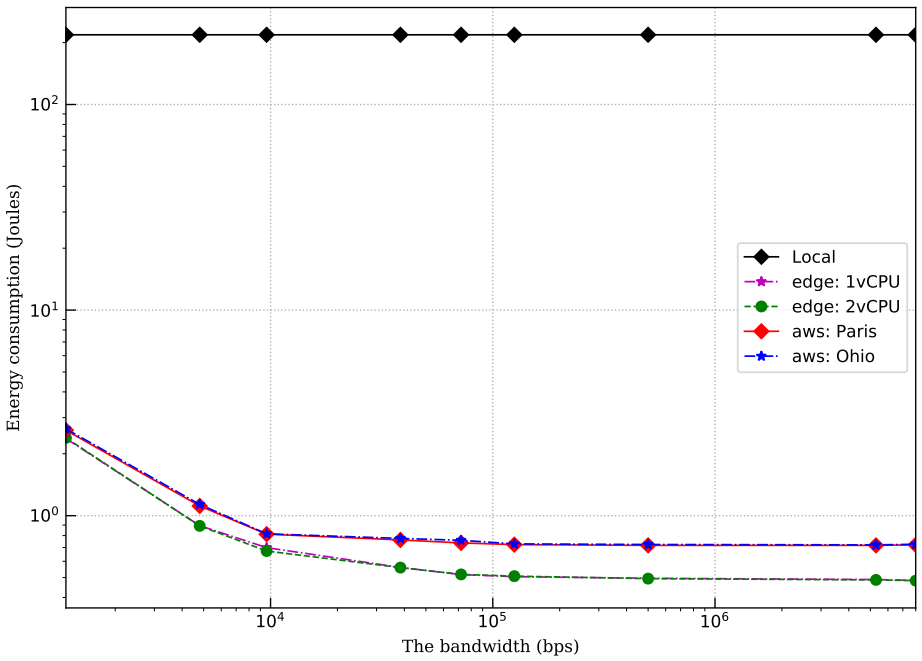



**Figure 4.14:** *Linpack full: Energy Consumption*

As shown in Figure 4.14, the observations made for Linpack *medium* are reinforced for the *full* variant. Because Linpack *full* is a CPU-intensive application, it is obvious that its completion time is mainly dominated by the computation duration. The transmission delay is proportionally negligible. This explains why the effect of differentiated RTT between Paris and Ohio is hardly observable in figure 4.14. However, the performance gap between the edge and the clouds is very apparent and almost insensitive to the bandwidth, when the latter reaches $72 Kbps$. We can deduce that the local edge offers

higher CPU capacity than those provided at the clouds, despite the fact that both have allocated 2vCPU resources. This discrepancy illustrates the impact of the heterogeneity of the hardware and the software technologies, which are used in a multi-tier mobile edge computing infrastructure.

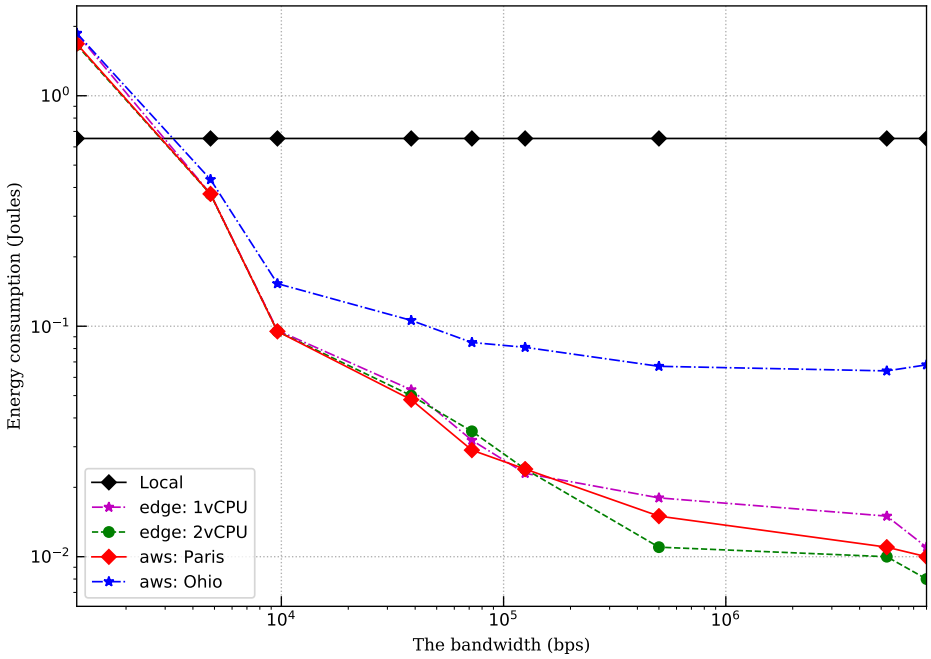



**Figure 4.15:** *CPUBench easy: Energy Consumption*

Energy consumption for CPU-Bench *easy*, *medium* and *full* are shown in figures 4.15, 4.16 and 4.17, respectively. The general observations regarding Linpack holds for CPUBench. However, for CPU-Bench *easy* one can see in figure 4.15 that when the bandwidth is restricted to less than $125Kbps$ the cloud in Paris and the local edge have almost the same performances, with a very slight advantage for the cloud at Paris. This can be explained by the fact that transmission duration over networks that exhibit important delays represent a significant proportion part with respect to the completion time of an offloaded light task. As shown in figure 4.13 when the bandwidth is restricted to less than $125Kbps$ the Round Trip Time from the terminal toward the local edge is elevated and quite similar to the RTT to the cloud at Paris.

Figures 4.18, 4.19 and 4.20 are relative to Pi-Bench. Compared to the previous bench applications one can notice for *easy* and *medium* cases that the edge with 2vCPU outperforms significantly the edge with 1vCPU. We believe that this is due to the

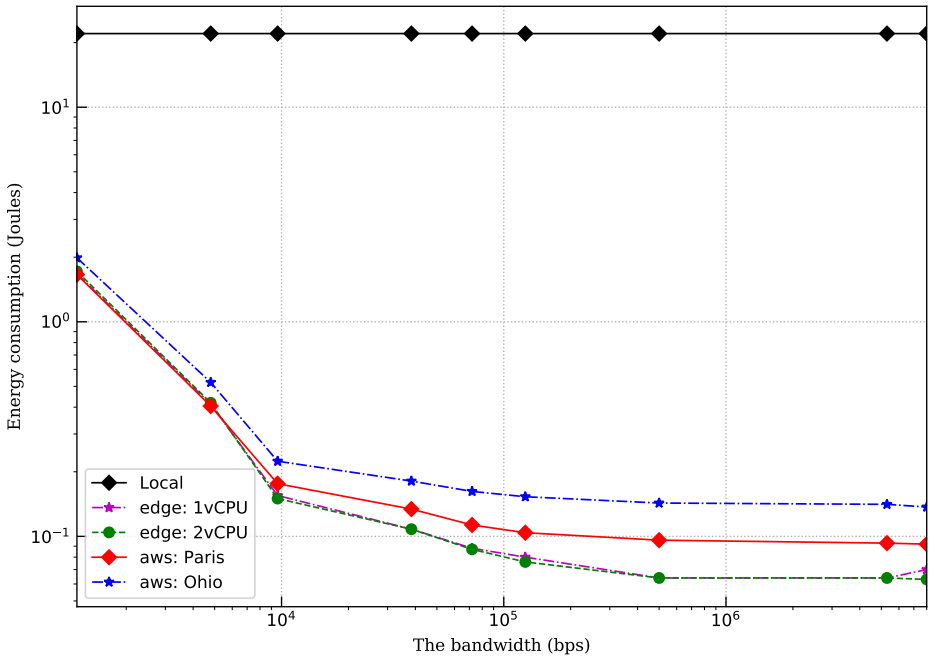



**Figure 4.16:** *CPUBench medium: Energy Consumption*

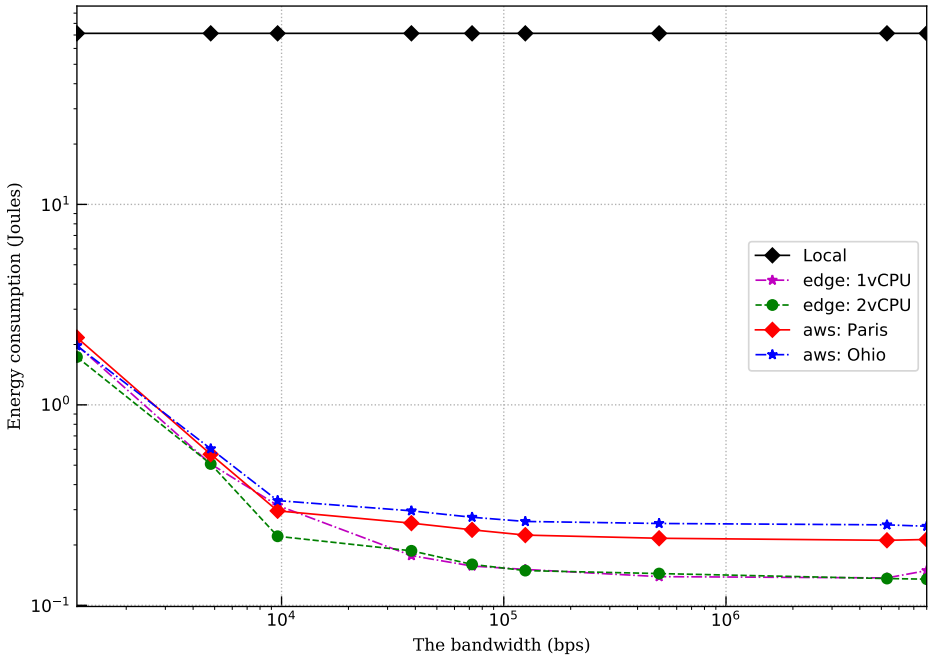



**Figure 4.17:** *CPUBench full: Energy Consumption*

multi-threaded nature of this application. The computation time is lower in 2vCPU compared to 1vCPU thanks to the ability of the former edge to process in parallel the computation of this multi-threaded application. The advantage of offloading such multithreaded application in a multi-CPU sever is clearly shown in figure 4.20 for the CPU-intensive case. Clouds in Paris and Ohio, where 2vCPU resources are allocated, outperform the 1vCPU edge.

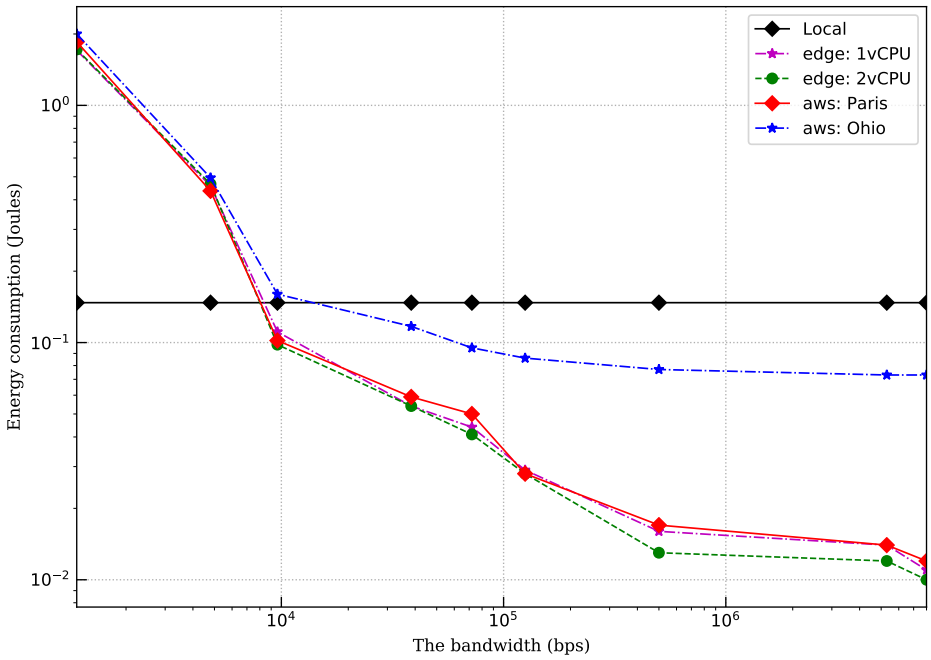



**Figure 4.18:** *PiBench easy: Energy Consumption*

Completion time for PiBench *easy*, *medium* and *full* are shown in Figures 4.21, 4.22 and 4.23, respectively. One can remark that this completion time curves have the same shape and follow the same hierarchy than those observed in energy consumption figures 4.19 and 4.20. Actually, in all our experiments we noticed that the consumed energy on the terminal is quasi-stationary during the execution time of an offloaded task. Consequently, the consumed energy is proportional to the completion time of an offloaded task. This observation holds for Linpack and CPU-Bench, as well[1]. The corollary to this finding is that in case of offloading, the remote location that minimizes the completion time is also the one that optimize the consumed energy. This statement has been validated in all the experiments that we have run, for the three bench applications,

[1]To save space, we choose not to show the completion time for Linpack and CPU-Bench

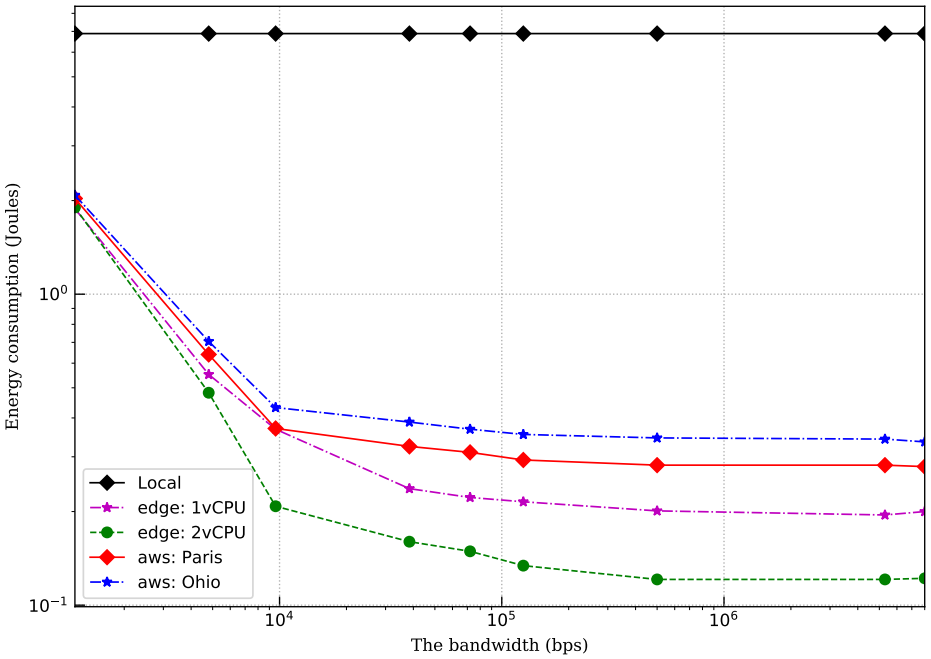



**Figure 4.19:** *PiBench medium: Energy Consumption*

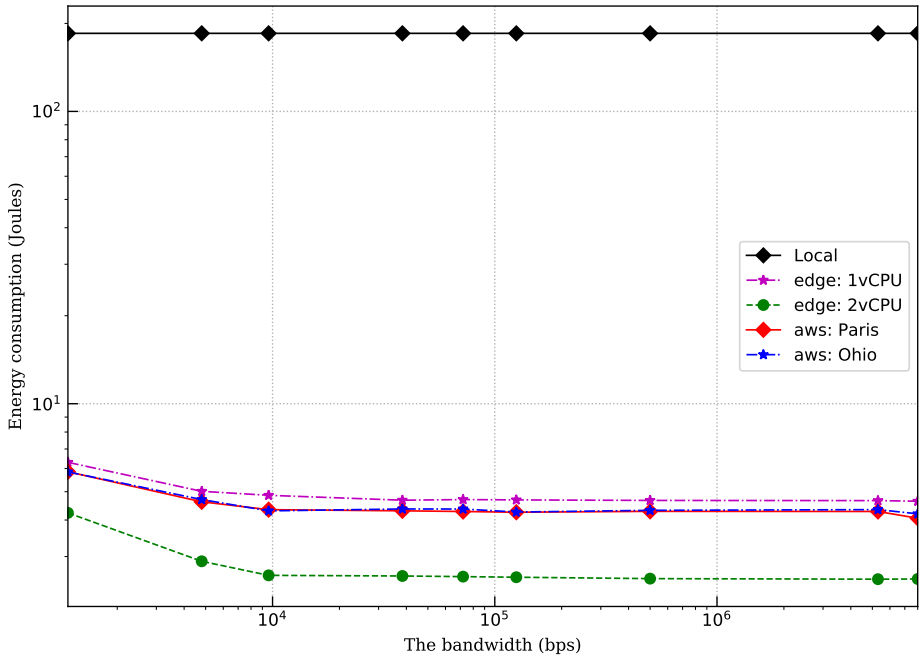



**Figure 4.20:** *PiBench full: Energy Consumption*

under different CPU and network configurations.

When a task is executed locally, the measured energy is also quasi-stationary during the processing time. However, as illustrated in figures 4.11, 4.12, 4.14, 4.15, 4.16, 4.17 4.18, 4.19 and 4.20 the difference of consumed energy between local and remote execution is of several order of magnitude. Yet, in almost all the experiments that we have run, we noticed that the decision among local and offloading that minimize the completion time is also the one that optimize the consumed energy. The unique observed exception case to this rule is Pi-Bench *easy* when the bandwidth is restricted to $9.6Kbps$. Comparing figure 4.21 to 4.18, we can remark that when the bandwidth is restricted to $9.6Kbps$ the local execution minimize the completion time, while energy consumption is minimized when Pi-Bench *easy* is offloaded on the 2vCPU edge.

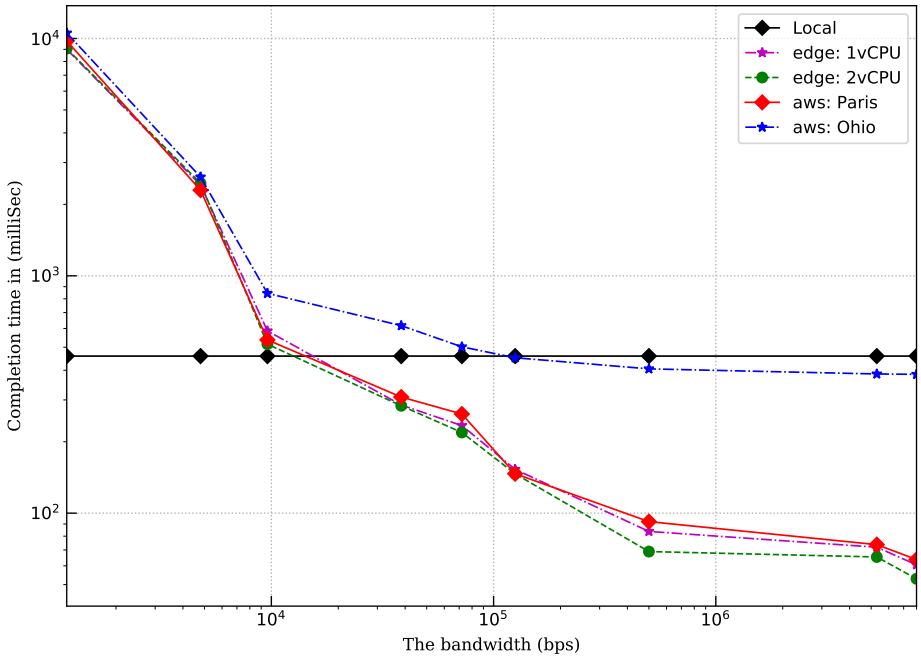



**Figure 4.21:** *PiBench easy: completion time*

### 4.5.3 Multi-user Scenario

In this section, we assessed the effect of multi-users on the offloading performances. We incrementally varied the number of users. However, due to the constraints on the available infrastructure nodes, at the time when we ran our experiments, we fixed the allocated resources both at the cloud and the edge to 1vCPU and we limited the number

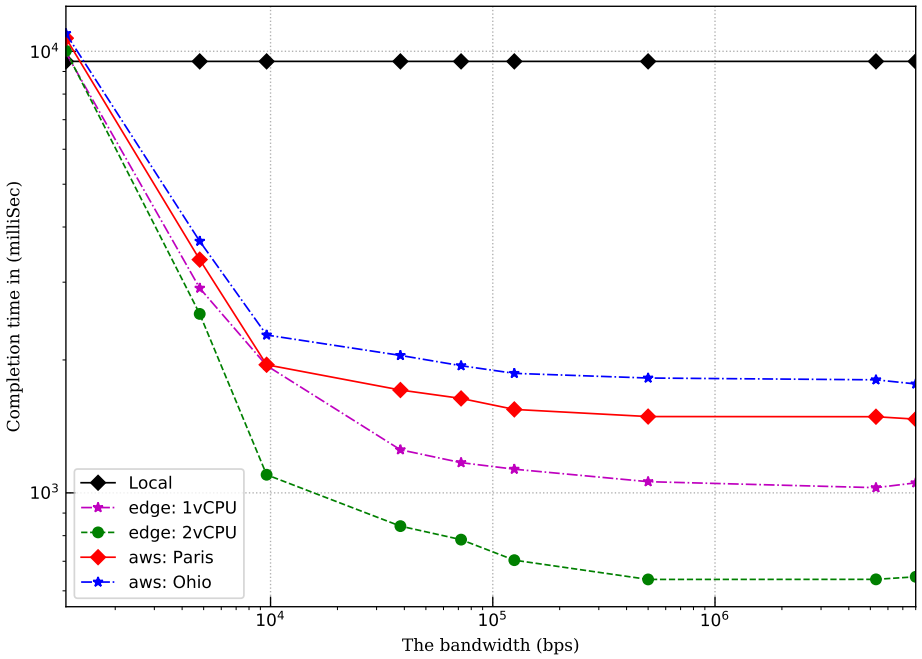



**Figure 4.22:** *PiBench medium: completion time*

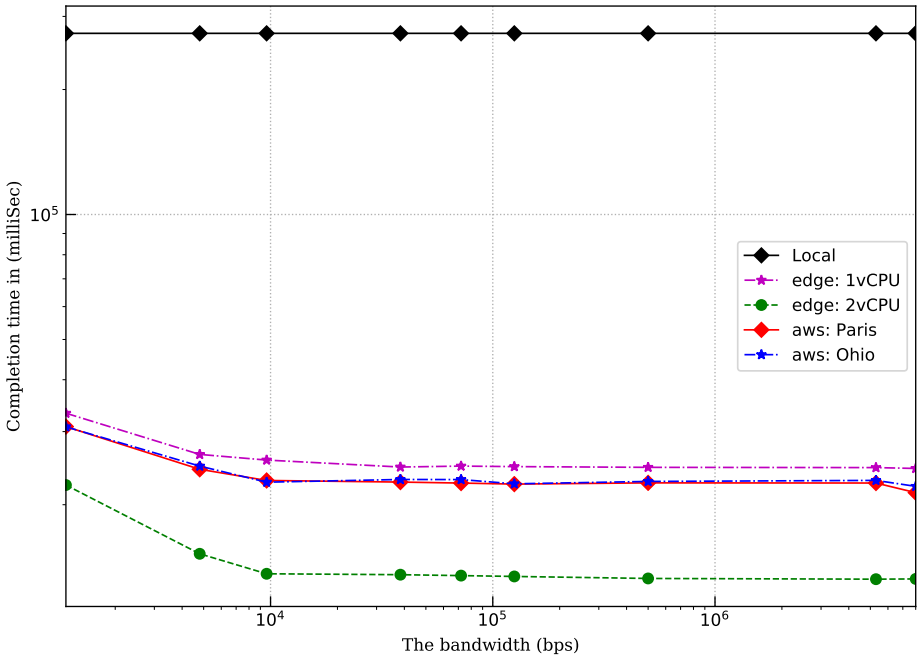



**Figure 4.23:** *PiBench full: completion time*

of mobile terminals to ten. Yet, we believe that the general tendencies, which we will comment hereafter, still hold for larger scale.

To save space, we also choose to discuss for the multi-users context case only the results of CPUBench. Our choice is motivated by the fact that following the one-user experiments results (see table 4.2)), CPUBench is the only application in our benchmark for which we observed three offloading decisions: local, edge and cloud.

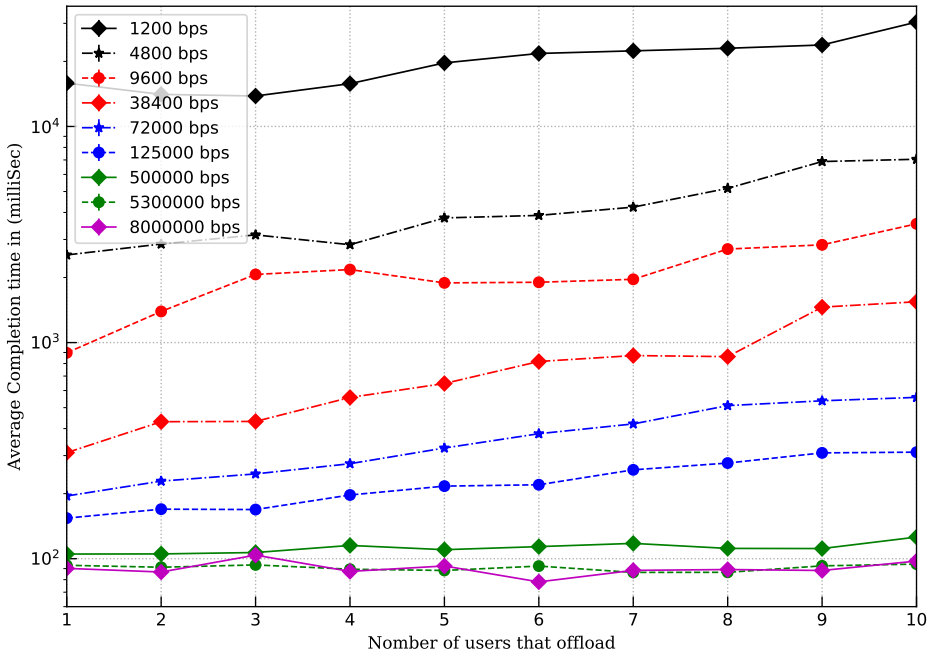



**Figure 4.24:** *Multi-users completion time of CPUBench easy in the cloud*

The completion time of CPU-Bench when it is offloaded on the cloud is represented in figures 4.24, 4.25, and 4.26, for *easy*, *medium* and *full* mode, respectively. Each curve in those figures is associated to an access bandwidth, between 1.2 kbps to 8Mbps. The $x$-axis represents the number of offloading users, while the $y$-axis represents in logarithmic scale the completion time in milliseconds. One can see that the completion time increases significantly with the number of users, especially for bandwidth capacities less than 500kbps. The contention among several users on the access network increases the transmission delay, especially when the bandwidth capacity is quite low. This effect is observed even for CPU-Bench in *full*, despite the fact the processing delay *versus* transmission delay is quite important for this mode. Compared to offloading on the edge, which is shown in figures 4.27 4.28 and 4.29, one can see that the completion time

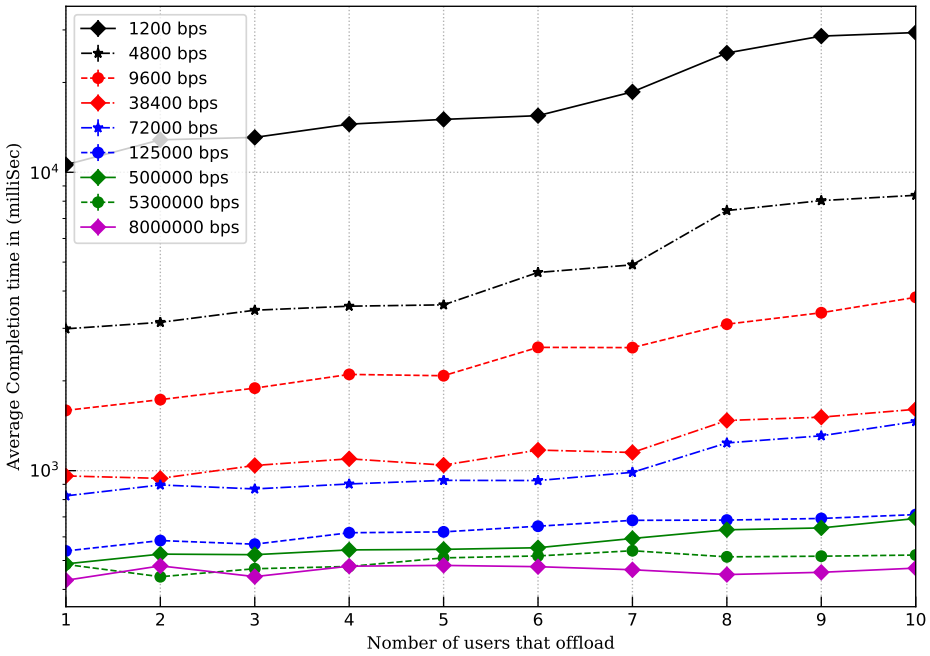



**Figure 4.25:** *Multi-users completion time of CPUBench medium in the cloud*

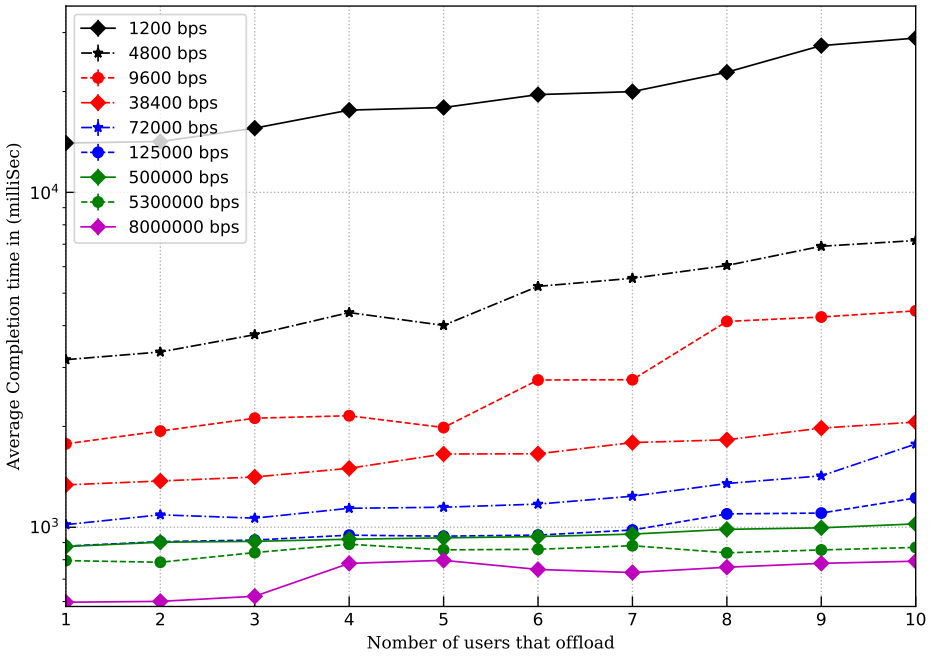



**Figure 4.26:** *Multi-users completion time of CPUBench full in the cloud*

in the edge increases slightly with the number of users, but the slope is much lower than for the cloud.

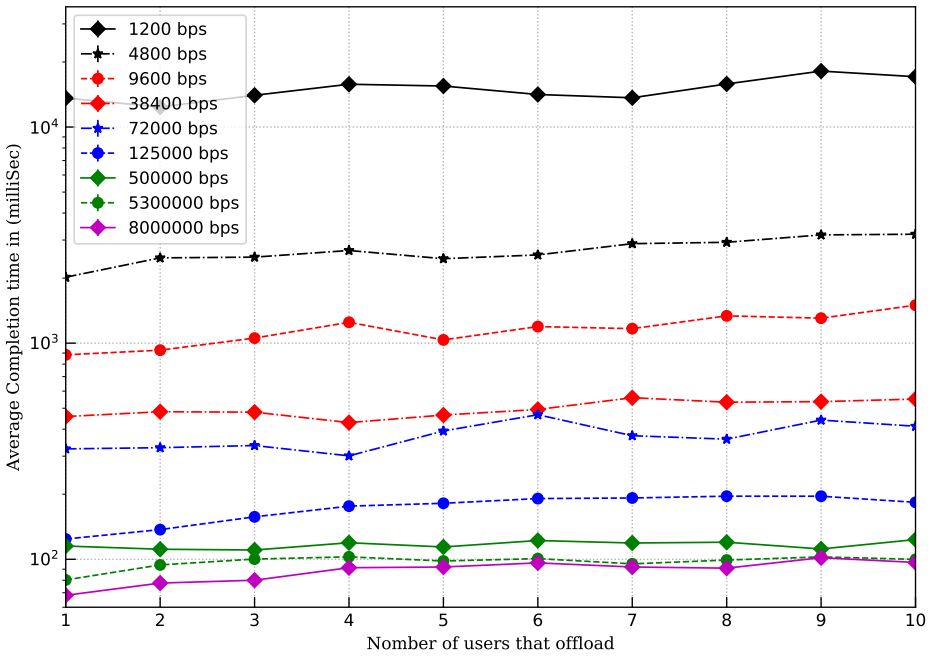



**Figure 4.27:** *Multi-users completion time of CPU-Bench easy in 1vCPU edge*

Completion time of 10 concurrent CPU-Bench users with local, edge and cloud execution locations is shown in figures 4.30, 4.31, and 4.32, for *easy*, *medium* and *full* mode, respectively. The $x$-axis represents in logarithmic scale the access bandwidth, which varies between 1.2 Kbps to 8Mbps, while the $y$-axis represents in logarithmic scale the completion time in milliseconds.

As shown in figure 4.30, local execution is the best placement for CPU-Bench in *easy* mode when bandwidth capacity is extremely constrained: less than 9,6Kbps. Indeed, in such condition, the transmission delay is very high and is proportionally much important than the processing delay, even when the application is executed in the edge or the cloud. Faster execution in the edge or in the cloud can not compensate the large transmission time when the access bandwidth is extremely low and shared among many (10) users. For such case, it is not worth it to offload and it is better to execute the task locally on the mobile terminal. For bandwidth capacity larger than 10Kbps, the experiments results show that the best decision is to offload on the edge. This result is interesting because it indicates that even when 10 users are contending on a quite

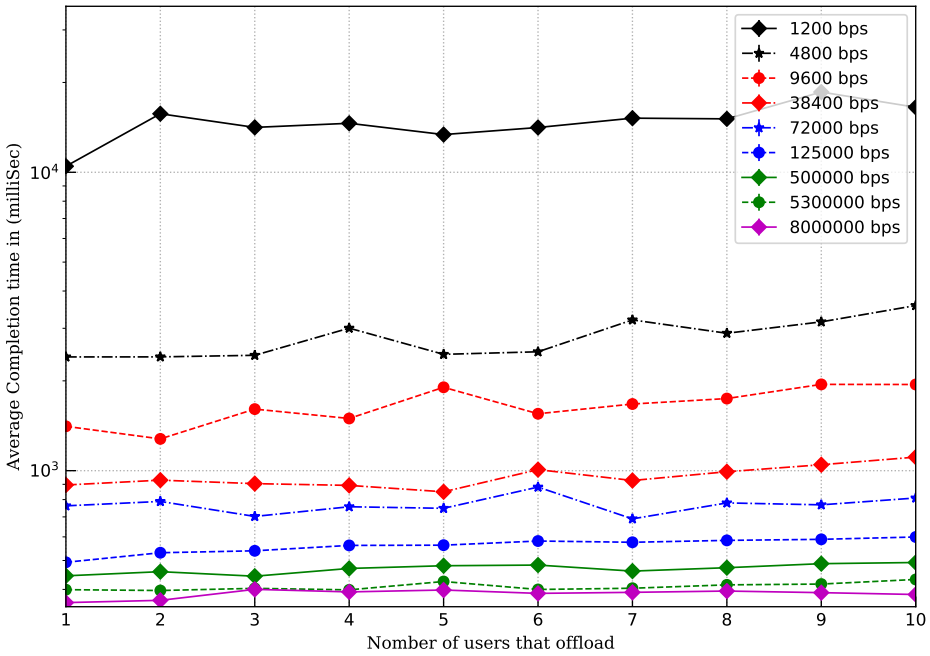



**Figure 4.28:** *Multi-users completion time of CPU-Bench medium in 1vCPU edge*

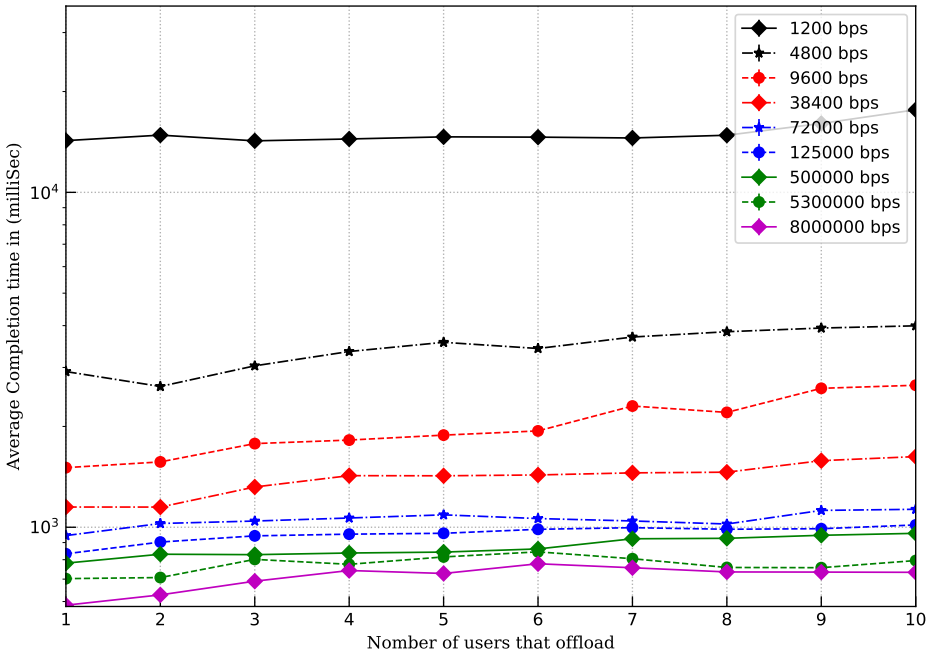



**Figure 4.29:** *Multi-users completion time of CPU-Bench full in 1vCPU edge*

limited bandwidth capacity of about 10Kbps, it still worth it to offload a computational intensive task such as CPU-Bench to the edge. Figures 4.30, 4.31, and 4.32 show that in almost all the cases the best offloading location is the edge. The only exception is for *easy* mode with a bandwidth access set to 5.3Mbps. As shown in figure 4.13, for higher bandwidth capacity (larger than few Mbps) the transmission delay to the edge is almost similar to the cloud. The difference is less than 10 milliseconds. In these cases, the completion time is almost the same on the edge and on the cloud, especially when the computational resources required by a task are low, which is the case for CPU-Bench in *easy* mode.

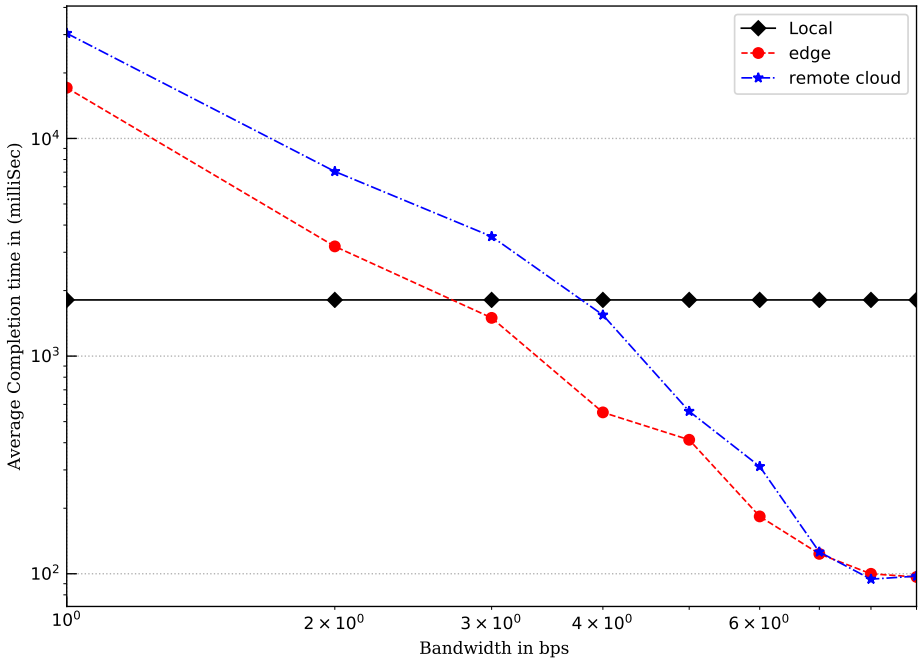



**Figure 4.30:** *10 users completion time of CPU-Bench easy*

## 4.6 Numerical Vs Experimental analysis

In this section, we compare and analyze the results obtained by experiments with those obtained numerically. In table 4.2 we summarize the results obtained for single-user scenario. The right column in table 4.2, shows the location that minimizes the consumed energy that we derived from experimental results for the three bench applications under different processing loads and network configurations. On the other hand, the left column indicates the optimal locations obtained by ECESO. To fit with the applications that

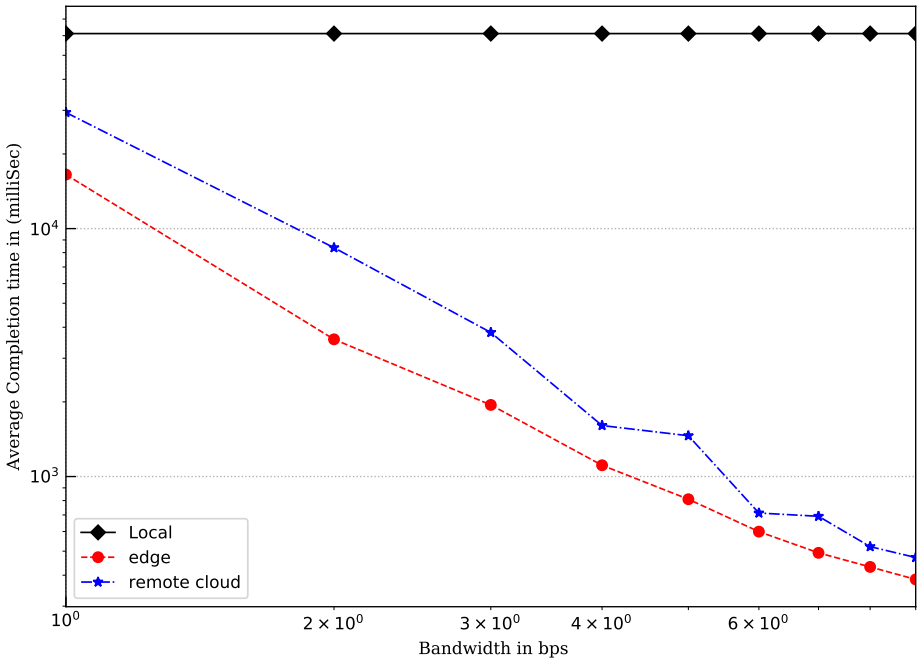



**Figure 4.31:** *10 users completion time of CPU-Bench medium*

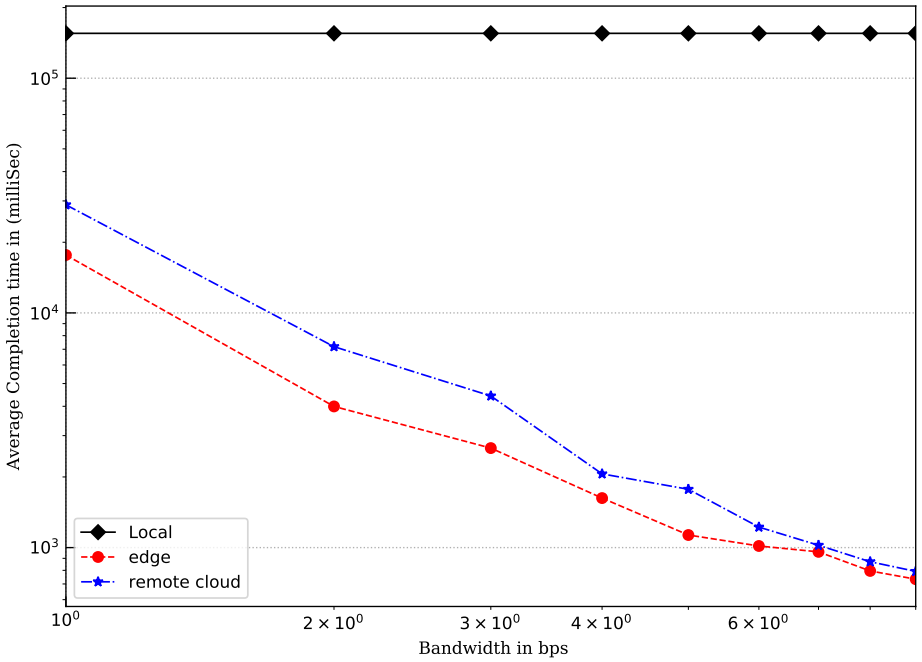



**Figure 4.32:** *10 users completion time of CPU-Bench full*

we evaluated in our testbed, we applied the measured parameters indicated in table 4.1 to ECESO. We can see that in the majority of the cases the theoretical solution match with the placement derived from the experimentation results. The few mismatches are indicated in red. For Linpack, the ECESO placement solution fits with the best placement observed *via* experiments in 22 cases among 27 tested configurations. Thus compared to experiments, our proposal have determined the optimal task execution location in 81.4% of the cases. To quantify the under-performance of ECESO in terms of consumed energy, we calculate the relative difference of the experimentally consumed energy among: 1) the case where task's computational location is computed by our proposal ECESO (left column in table 4.2) 2) the case where optimal placement solution is derived from experiments observations (right column in table 4.2). Averaging over the 27 possible cases, the under-performance of ECESO in terms of consumed energy with respect to an optimal placement derived from experimental results, is limited to 1.962%.

The near-optimality of ECESO, is also confirmed for Pi-Bench and CPU-Bench. For Pi-Bench, the optimal placement is achieved by ECESO in 85.18% of the studied cases, with an average under-performance of 2.57% of consumed energy. For CPU-Bench optimal placement is obtained by ECESO in 70.37% of the cases with an average under-performance of 3.76% of consumed energy.

Finally, in table 4.3, we compare the best offloading decisions of CPU-Bench obtained by experimentation and using ECESO, in the case of multi-user scenario (10 users). Each line is associated to a given bandwidth capacity at the access network. Thus combining with the three modes of CPU Bench, we have in total 27 cases. Table 4.3 shows that the decision of ECESO matches with the best placement observed through experiments in 66.7% of the cases. The observed errors are due to the conservative nature of our algorithm. For example, for extremely low bandwidth capacity (low than 3.2 Kbps) ECESO recommends to execute the task locally rather than offload it to the edge. Similarly, for *easy* mode with 5.3 Mbps bandwidth access case that we discussed above, our algorithm suggests to offload the task on the edge, while experiments have shown that offloading on the cloud can also minimize the completion time. The conservative nature of ECESO is mainly inherent to our modelling, which induces some assumptions that appears to be conservative compared to real system behaviors, as observed in experiments.

**Table 4.2:** *Offloading decisions for each application: ECESO vs Experiments*

| Bandwidth | ECESO | | | Experimentation | | |
|---|---|---|---|---|---|---|
| | Easy | Medium | Full | Easy | Medium | Full |
| | | | Linpack | | | |
| 1 200 | Local | 2vCPU | 2vCPU | Local | 1vCPU | 1vCPU |
| 4 800 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 9 600 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 38 400 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 72 000 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 1vCPU |
| 125 000 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 1vCPU |
| 500 000 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 5 300 000 | Local | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 8 000 000 | Local | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| | | | PiBench | | | |
| 1 200 | Local | 2vCPU | 2vCPU | Local | 1vCPU | 2vCPU |
| 4 800 | 2vCPU | 2vCPU | 2vCPU | Local | 2vCPU | 2vCPU |
| 9 600 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 38 400 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 72 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 125 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 500 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 5 300 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 8 000 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| | | | CPUBench | | | |
| 1 200 | 2vCPU | 2vCPU | 2vCPU | Local | Paris | 2vCPU |
| 4 800 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | Paris | 2vCPU |
| 9 600 | 2vCPU | 2vCPU | 2vCPU | Paris | 2vCPU | 2vCPU |
| 38 400 | 2vCPU | 2vCPU | 2vCPU | Paris | 1vCPU | 1vCPU |
| 72 000 | 2vCPU | 2vCPU | 2vCPU | Paris | 2vCPU | 2vCPU |
| 125 000 | 2vCPU | 2vCPU | 2vCPU | 1vCPU | 2vCPU | 2vCPU |
| 500 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 1vCPU | 1vCPU |
| 5 300 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |
| 8 000 000 | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU | 2vCPU |

**Table 4.3:** *Offloading decision for CPUBench with 10 users: ECESO vs Experiments*

| Bandwidth | ECESO | | | Experimentation | | |
|---|---|---|---|---|---|---|
| | Easy | Medium | Full | Easy | Medium | Full |
| 1 200 | *Local* | *Local* | *Local* | *Local* | *1vCPU* | *1vCPU* |
| 4 800 | *Local* | *Local* | *Local* | *Local* | *1vCPU* | *1vCPU* |
| 9 600 | *Local* | *Local* | *Local* | *1vCPU* | *1vCPU* | *1vCPU* |
| 38 400 | *Local* | *Local* | *Local* | *1vCPU* | *1vCPU* | *1vCPU* |
| 72 000 | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* |
| 125 000 | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* |
| 500 000 | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* |
| 5 300 000 | *1vCPU* | *1vCPU* | *1vCPU* | *Paris* | *1vCPU* | *1vCPU* |
| 8 000 000 | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* | *1vCPU* |

## 4.7 Conclusion

In this chapter, we study the extension of D2M-ECOP offloading approach in two-tier MEC environment. The extended offloading policy, ECESO, considers all the available computing resource in the MEC (edge servers and the remote cloud) and selects the best place to perform the offloaded tasks. The numerical results show the importance of considering two-tier MEC on the performance of the computation offloading for multi-user scenario. In addition, we implemented ECESO in a real-world testbed. Using this testbed, we evaluate our offloading decision policy for three real Android OS applications, with different traffic patterns, resource demands, and multi-users context. The experimental results highlight the efficiency of ECESO computation offloading policy. Moreover, they illustrate that the nature of how the application is developed affects highly the performance of the offloading.

However, ECESO supposes that the offloadable task of an application is always known and determined statically at the design time. The computation task to offload is always the same. However, in a real-world system the resources maybe not enough to offload the hole computation of the task, or there is more resource than the required by that task. In both cases, ECESO does not take on consideration to reduce or increase the computation to offload, which affects the performance of offloading. To deal with such a situation, we focus on the multitasking application in the next chapter.

CHAPTER 5

# Elastic Offloading of Multitasking Applications to MEC

*eTOMEC offloading policy*

**Abstract**

This chapter focuses on offloading multitasking application to a MEC environment. It address this issue through a mobile user's perspective that is seeking to obtain the execution result of a resource-hungry multitasking application, possibly through offloading some tasks to a multi-edge servers MEC. The completion time of the application is constrained by a predefined strict deadline and the offloading decision aims to minimize the mobile terminal's energy consumption. The numerical results shows that the proposed offloading policy *eTOMEC* achieves better performance than existing solution [80].

[80] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Elastic Offloading of Multitasking Applications to Mobile Edge Computing". In: *Proceedings of the 22st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2019, Miami Beach, FL, USA, November 25 - November 29, 2019*. 2019, pp. 1–8. DOI: 10.1145/3345768.3355926

## 5.1 Introduction

Computation offloading has been addressed by several recent works [49]. Most of this works focused on the the offload decision, which primary aims to determine when to offload and to decide whether the application execution should be completely or partially offloaded to the MEC environment. In addition, they investigated the choice of the corresponding execution location, including either the mobile terminal or to a set of edge or cloud servers. As presented in the previous chapters 3 and 4, the existing offloading approaches consider the optimization of performance metrics such as the completion time and the energy consumption of the application [33, 68, 69]. Moreover, these approaches are based on single-task architecture type of the application where the computation to offload is determined in advance, and it is always the same whatever the conditions at the runtime. While some approaches have investigated multitasking applications, the dependencies between tasks are often overlooked.

To tackle this limitation, in this chapter we explore the computation offloading for a multitasking application characterized by a task-dependency graph to a multi-edge server MEC environment. Our objective is to design an offloading approach that is able to scale the number of concurrent offloaded tasks accordingly with the number of available edge servers, while taking into consideration tasks dependencies. The offloading aims to minimize the mobile terminal's energy consumption, while satisfying the hard deadline of the application's completion time. The design of such offloading approach is quite challenging, since it must integrate on one side, tasks dependencies and their resource requirements (processing and communication capacities) and on the other side, available resources on different servers and communication aspects (available bandwidth, delay and energy costs) at access and at backbone links.

The first contribution of this chapter is the formalization of the above optimal offloading problem as a Zero-one Integer Programming problem. Given that the resulting problem is NP-hard, we then propose a heuristic solution, named eTOMEC. In first, eTOMEC starts by selecting the tasks that should be offloaded according to a priority corresponding to the energy cost in the case of the worst case offloading decision. Once the task selected, it selects the offloading server that minimize the extra amount of energy that could be consumed by the user's terminal when scheduling the selected task to that server. Finally, in the case where the obtained solution is not satisfying the hard deadline of the application's completion time, we propose a merging step that

merges tasks that reduce the completion time.

The rest of the chapter is organized as follows: section 5.2 describes the studied system. Multitasking offloading problem is formulated in section 5.3. Our proposed offloading policy, named *eTOMEC*, is then developed in section 5.4. Performance evaluation results are analyzed in section 5.5. Finally, a conclusion is drawn in section 5.6.

## 5.2 System description

In this section, we describe the MEC infrastructure. Then, we introduce the considered application's model and we detail the task-dependency graph of the application.

### 5.2.1 MEC infrastructure

As illustrated in Figure 5.1, we consider a multi-tier MEC environment, which is composed of several physical or virtual machines offering computing resources at the edge and at the cloud. We study this system through a single user's perspective, which is connected to this environment through an AP and that is seeking to execute a computationally intensive application. This application is composed of a set of inter-dependent tasks. Each task can be performed locally by the user terminal, offloaded to edge servers or offloaded further to the remote cloud.

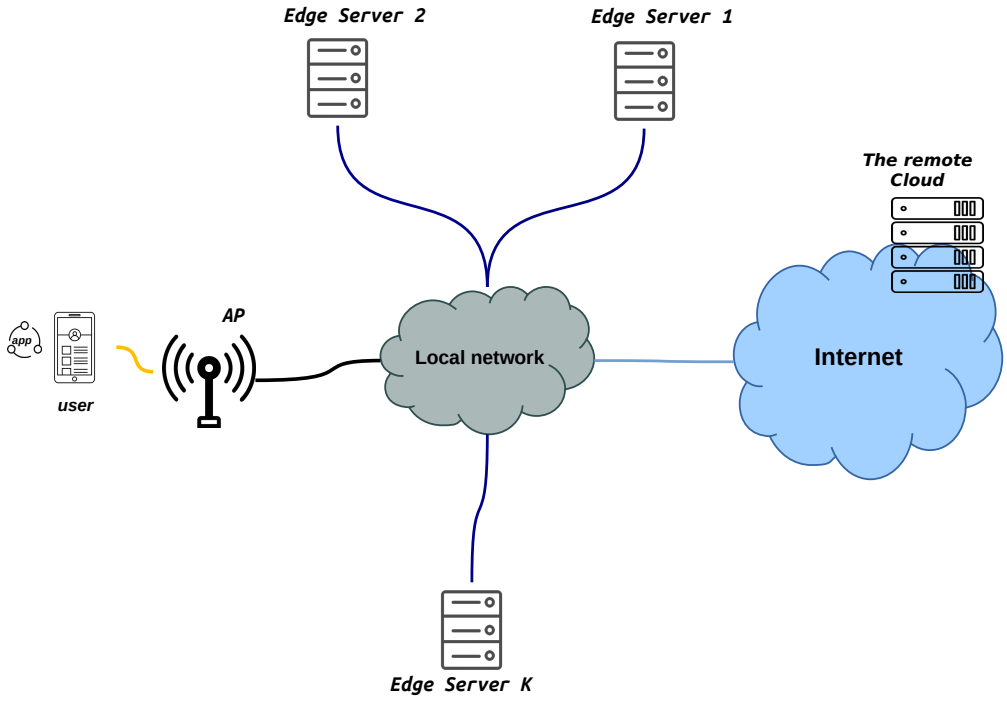



**Figure 5.1:** *An illustration of MEC infrastructure environment.*

Let $\mathscr{K} = \{0,1,2,...,K,K+1\}$ denotes the set of all possible processing locations of tasks. Here, the user terminal is referred by 0, the edge servers represented by 1 to $K$ and $K+1$ represents the remote cloud. In the remainder, each element of $\mathscr{K}$ is referred as *server*. Each server offers a computational resource capacity, denoted $f^k$, and expressed as a CPU-cycles per second. Similarly, to [33, 49], we assume that the user terminal has a limited computing capabilities, compared to the edge/cloud. Formally, $\forall k \in \mathscr{K} \setminus \{0\},\ f^0 \ll f^k$.

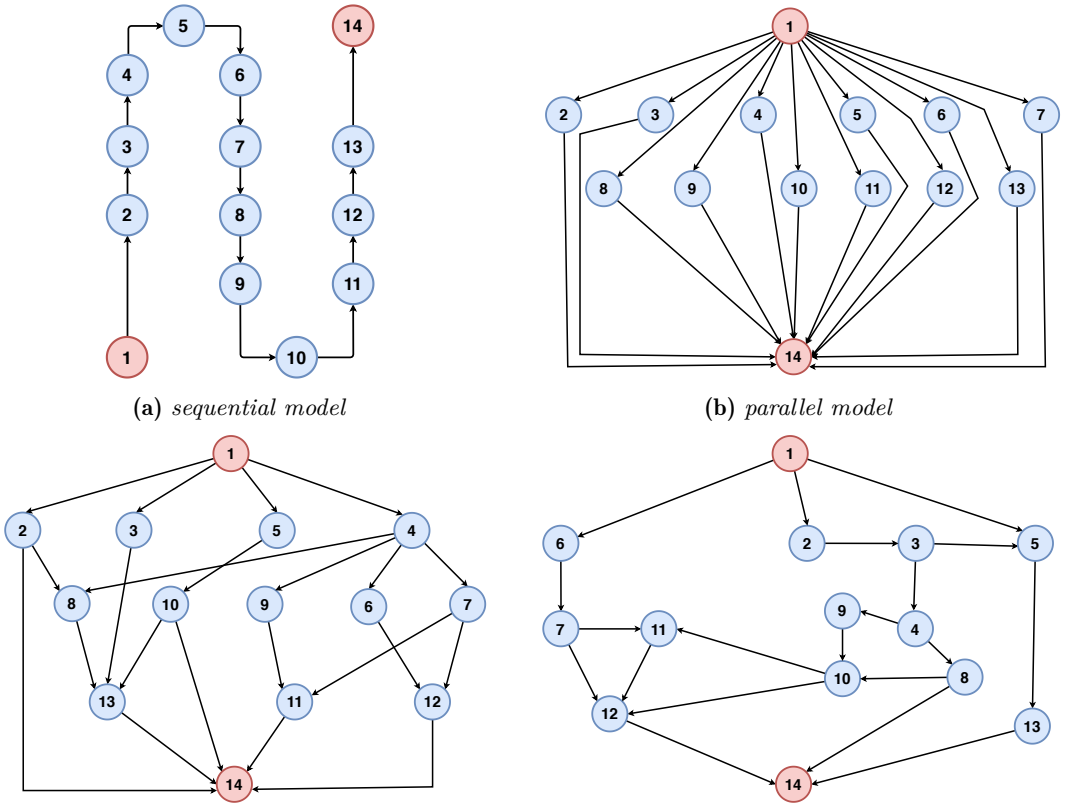
### 5.2.2 Multitasking application modeling

We consider a mobile user that is seeking to obtain the execution result of a resource-hungry multitasking application by offloading some tasks to a MEC environment. The offloading decision aims to minimize the terminal's energy consumption while ensuring that the completion time of the application is constrained by a predefined strict deadline.

The multitasking application is composed of a set of fine granularity atomic non-preemptive tasks. The dependencies between tasks is modeled using a weighted Directed Acyclic Graph (DAG), noted $G = (V,E)$, where the vertices in $V$ represent tasks and the edges in $E$ represent dependencies. In the following, we will refer to the number of tasks composing the application as $n = |V|$.

The construction of such multitasking DAG depends on how the developers model their applications [33, 68]. Hence, several DAG could be associated to different implementations of a given multitasking application. For instance, figure 5.2 illustrates four DAG that correspond to different implementations on Android terminals of a video navigation application [47]. In this figure, each edge illustrates the precedence dependencies between tasks. Concretely, any edge $(i,j) \in E$ means that task $i$ must be performed before task $j$. In the following, we will use $pred(i) = \{j|(j,i) \in E\}$ to refer to the immediate predecessors of a vertex $i \in V$ and $succ(i) = \{j|(i,j) \in E\}$ to represent the immediate successors of $i$ in $G$.

We can also distinguish in figure 5.2, two type of vertexes: blue and red ones. The latters are either (a) a vertex without any predecessors, called *entry task* (task 1), from which the application execution begins or (b) a vertex without any successors called *exit task* (task 14), which is the latest task executed by the application. Since these tasks are usually related to users interactions (e.g, user inputs and results rendering), we assume that both *entry* and *exit* tasks must be performed on the user's terminal. Such




**(a)** *sequential model* **(b)** *parallel model*

**(c)** *layer by layer model* **(d)** *Fan In/Fan Out model*

**Figure 5.2:** *The DAG associated to video navigation application [47].*

local execution requirement can be extended to other tasks, which cannot be performed remotely due to some hardware or software constraints [31, 69]. To identify those tasks we introduce an *non-offloadability indicator*, denoted by $y_i$, which is set to 0 if task $i$ can be offloaded on a remote server, and set to 1 if it must be processed on the mobile's terminal. In particular, we have $y_1 = 1$ and $y_n = 1$.

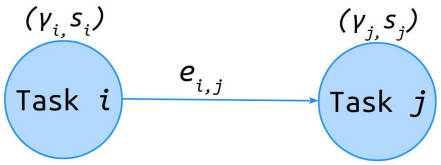



**Figure 5.3:** *Example of dependent tasks*

In addition to the precedence order expressed by each edge, we associate to the edge $(i, j) \in E$ a weight, denoted by $e_{i,j}$. This weight expressed in bytes, indicates the amount of task $i$ resulting data that will serve as input data to process task $j$. A generic example of this weighted graph is illustrated in figure 5.3. As shown in this figure, a weight $(\gamma_i\ , s_i)$ is also associated to each vertex $i \in V$. Here, $\gamma_i$ denotes the computation capacity, in CPU-cycles, which is required to process task $i$. The second parameter,

$s_i$, is the source code size, in bytes, which needs to be transferred to a remote server when task $i$ is offloaded. Hence, the weights on a vertex $i$ and on its entering edges characterize the resources (processing capacity and data quantity) needed to process task $i$.

## 5.3 Multitasking offloading problem

In this section, we present the computation offloading problem for multitasking application in multi-edge server MEC environment. Then, we define completion time and the energy consumption of the application.

### 5.3.1 Problem statement

Our purpose is to design an offloading policy that decides 1) which tasks should be offloaded and 2) determines which server will process each offloaded task. The objective of this policy is to minimize the energy consumed by the user terminal, denoted as $\mathscr{Z}$. Moreover, the offloading decisions must fulfill the following requirements: First of all, each task needs to be scheduled onto exactly one server with respect to the precedence constraints, which is given by the weighted Directed Acyclic Graph associated to the multitasking application. Second, the offloading policy must ensure that the maximum completion time of the application, denoted by $\mathscr{T}$, does not exceed a given threshold, denoted $t_{max}$. This hard constraint is introduced with the aim to enforce Quality of Services requirements of time-sensitive applications. Finally, non-offloadable tasks must be performed locally by the user terminal. Formally, the optimal offloading policy is the solution of the following Zero-one Integer Programming problem:

$$
\begin{aligned}
&\textbf{Minimize } \mathscr{Z} \\
&\textbf{Subject to:} \\
&C1 : \mathscr{T} \leq t_{max} \\
&C2 : \sum_{k \in \mathscr{K}} x_i^k = 1, \forall i \in V \\
&C3 : 1 - x_i^0 \leq y_i, \forall i \in V \\
&C4 : x_i^k \in \{0,1\}
\end{aligned}
\tag{5.1}
$$

Here, $x_i^k$ as a binary variable that indicates if task $i \in V$ is offloaded to server $k$. The first constraint $(C1)$ guarantees that the application's completion-time, $\mathscr{T}$, is below the given deadline $t_{max}$. Constraint $C2$ indicates that each task must be performed by exactly one server. Constraint $C3$ is relative to the fact that the non-offloadable tasks must be performed at the user's terminal. Finally, constraint $C4$ ensures that every decision variable is binary.

In the following, we will detail the expressions of the objective function $\mathscr{Z}$ and the constrained application's completion-time $\mathscr{T}$. To this purpose, we will first describe the network communication model. Then we will develop different time-related metrics that will be used to model the application's completion time and to derive the expression of energy consumption.

### 5.3.2 Completion time

Clearly, the completion of the application is obtained when the *exit* task is processed. Thus, the expression of $\mathscr{T}$ is composed of the processing time of all application's tasks, plus different transmission times. Indeed, communications occur at different steps of the offloading process. Precisely, whenever a task is offloaded to the MEC environment, its source code is transmitted from the user terminal to the remote server. Moreover, dependencies among tasks lead to data exchanges among different locations processing the tasks. Hence, to derive the expression of $\mathscr{T}$ we will first characterize the bandwidth at access and at backhaul networks.

#### 5.3.2.1 Bandwidth capacity

Let first focus on the allocated wireless bandwidth at access network. To this purpose, we suppose that the AP has the same configurations used in the section 3.2.6 of the chapter 3, which is the orthogonal frequency division multiple-access (OFDMA) technology.

Let $B$ denote the available wireless bandwidth in the AP. This quantity is expressed in Hertz (Hz). Since we are considering OFDMA, $B$ is divided into $\mathcal{B}$ subcarriers. We define $p^{tx}$ as the user's transmission power, and $h^{tx}$ as the power gain. We consider both path loss and shadowing attenuation as in [60]. We model the noise as *Additive White Gaussian Noise (AWGN)* [60] random variable with zero mean and variance $\sigma^2$.

Thus, the maximum achievable rate (in bps) for the uplink can be expressed as follows:

$$w^{tx} = \upsilon \cdot \frac{B}{\mathcal{B}} \cdot log_2(1 + \frac{p^{tx} \cdot (h^{tx})^2}{\Gamma(b^{tx}) \cdot d^2 \cdot \sigma^2}) \tag{5.2}$$

where $\Gamma(BER)$ represents the SNR margin introduced to meet the desired target bit error rate (BER) with a QAM constellation, $b^{tx}$ is the BER,$d$ is the distance between the user and the AP, and $\upsilon$ the number of subcarriers allocated to the user. $\Gamma(BER)$ can be expressed as follows:

$$\Gamma(x) = -2 \cdot \frac{log(5x)}{3} \tag{5.3}$$

Similarly, the maximum achievable bitrate (in bps) for downlink can be derived by the following equation:

$$w^{rx} = \upsilon \cdot \frac{B}{\mathcal{B}} \cdot log_2(1 + \frac{p^{ap} \cdot (h^{rx})^2}{\Gamma(b^{rx}) \cdot d^2 \cdot \sigma^2}) \tag{5.4}$$

where $p_{ap}$ denotes the transmission power of the AP. $h^{rx}$ indicates the power gain from the user to the AP when user receives data due to path loss and shadowing attenuation [60].

Finally, the bandwidth capacity between any server $m \in \mathscr{K}$ and any server $k \in \mathscr{K}$, denoted $W^{m,k}$, is supposed to be symmetric and fixed during the application execution period. Notice that when $m = 0$ then $W^{0,k}$ refers to the bandwidth between the access point and a server $k \in \mathscr{K} - 0$.

#### 5.3.2.2 Transmission time of source code

Using equation 5.2, we can derive the time required to transmit the source code of task $i$ from the user terminal to server $k$. This time, denoted $sr_i^k$, can be expressed as the ratio between the amount of source code ($s_i$) and the available uplink bandwidth between user $i$ and server $k$. Formally:

$$sr_i^k = \begin{cases} 0 & \text{if } k{=}0 \\ \frac{s_i}{w^{tx}} + \frac{s_i}{W^{0,k}} & \text{otherwise} \end{cases} \tag{5.5}$$

#### 5.3.2.3 Transmission time of input data from predecessor tasks

As mentioned previously, the input data of task $i$ is the amount of all data communicated from its predecessor tasks. Let j $\in pred(i)$ be a predecessor of task $i$ and suppose that task $j$ has been performed by server $m$, then the time required to transmit the depending on data between tasks $j$ and $i$ from server $m$ to server $k$. This time, denoted by $\mathcal{D}_{j,i}^{m,k}$, must be expressed as a function of the amount of data, $e_{j,i}$, the location of the servers $m$ and $k$, and the available network bandwidth between server $m$ and $k$. We can distinguish four cases:

i. When task $j$ is performed locally ($m = 0$) and $i$ is performed remotely ($k > 0$), then $\mathcal{D}_{j,i}^{m,k}$ corresponds to the upload time of data dependency between user terminal and server $k$.

ii. When task $j$ is performed remotely and task $i$ is going to be performed locally, i.e: $k = 0, m > 0$, then $\mathcal{D}_{j,i}^{m,k}$ corresponds to the download time of the data dependency.

iii. When both tasks $i$ and $j$ will be performed by remote servers, i.e: $m > 0, k > 0$, then $\mathcal{D}_{j,i}^{m,k}$ corresponds to the backhaul transmission time between servers $m$ and $k$.

iv. Finally, when both tasks $j$ and $i$ are going to been executed by the same server then there is no communication, and $\mathcal{D}_{j,i}^{m,k} = 0$.

Therefore, using equations 5.2 and 5.4, we can obtain the following expression:

$$\mathcal{D}_{j,i}^{m,k} = \begin{cases} \frac{e_{j,i}}{w^{tx}} + \frac{e_{j,i}}{W^{m,k}} & \text{if } m = 0 \text{ and } k \neq 0 \\ \frac{e_{j,i}}{w^{rx}} + \frac{e_{j,i}}{W^{m,k}} & \text{if } m \neq 0 \text{ and } k = 0 \\ \frac{e_{j,i}}{W^{m,k}} & \text{if } (m \neq k) \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

#### 5.3.2.4 Task processing time

The task $i \in V$ can be processed by a server $k \in \mathcal{K}$ if enough processing resources are available at server $k$ and when both input data and source code of the task have been

received by server $k$. Let $st_i^k$ denote the time at which the processing of the task $i$ in server $k$ can start. This time can be expressed as follows:

$$st_i^k = max(com_i^k, av^k, sr_i^k) \tag{5.7}$$

where $com_i^k$ is the time required to receive all the input data of task $i$. $av^k$ refers to the time at which to server $k$ is ready to perform task $i$, and $sr_i^k$ is the time to transmit the source code of task $i$ to server $k$, which is expressed by equation 5.5. Following [33], $com_i^k$ can be defined as the last arrived data dependency from the predecessors tasks, and defined as follows:

$$com_i^k = \underset{m \in \mathscr{K}}{max}( \underset{j \in pred(i)}{max} [x_j^m \cdot (ft_j^m + \mathcal{D}_{j,i}^{m,k})]) \tag{5.8}$$

where $ft_j^m$ is the finish time of the task $j$ in the server $m$. The time to transmit dependency data, $\mathcal{D}_{j,i}^{m,k}$, is given by equation 5.6. It is worth noting here the presence of the binary offloading decision variable, $x_j^m$ in the expression of $com_i^k$. Similarly, the $av_i^k$ can be expressed as follows:

$$av^k = \underset{i \in V}{max}(x_i^k \cdot ft_i^k) \tag{5.9}$$

The processing time of task $i$ in serve $k$, denoted by $T_i^k$, can be estimated as the ratio between the required and the allocated computing resource in the server [43, 68]:

$$T_i^k = \frac{\gamma_i}{f^k}, \forall k \in \mathscr{K} \tag{5.10}$$

The finish time of the task $i \in V$ in server $k$, denoted $ft_i^k$, is defined as the summation of start time $st_i^k$ and the processing time $T_i^k$. Formally:

$$ft_i^k = st_i^k + T_i^k \tag{5.11}$$

Finally, since application's completion corresponds to the finish time of *exit* task and given that the later is non-offloadable, we can thus deduce that:

$$\mathscr{T} = ft_n^0 \tag{5.12}$$

### 5.3.3 Consumed energy

Let now express our objective function, which is the energy consumption of the user's terminal. This quantity is composed of four elements

i. The energy consumed by the local CPU due to local processing of tasks.

ii. The energy consumed by the wireless network interface when uploading to remote servers the source code, and the data of the offloaded tasks.

iii. The energy consumed by the wireless network interface when downloading tasks execution results from remote servers.

iv. Finally, the energy consumed by the wireless network interface when it is in idle mode. This mode is enabled when user terminal is waiting for the execution of those offloaded tasks.

Using the model presented in [37, 68] and according to the last considerations, we can compute the total energy consumption, noted $\mathscr{Z}$, as follows:

$$\mathscr{Z} = \sum_{i \in V} (\mathscr{E}_i^l + \mathscr{E}_i^{tx} + \mathscr{E}_i^{rx}) + \mathscr{E}^{idle} \tag{5.13}$$

where $\mathscr{E}_i^l$ is the energy consumed by the user's terminal to process task $i$ locally. Following [61], this energy is related to the local CPU frequency and the execution duration of the task. Formally:

$$\mathscr{E}_i^l = \kappa \cdot (f^0)^3 \cdot T_i^0 \cdot x_i^0 \tag{5.14}$$

Here $\kappa$ is the effective switched capacitance, which depends on the chip architecture, and is used to adjust the processor frequency. As in [61], we set $\kappa = 10^{-9}$.

$\mathscr{E}_i^{tx}$ represents the energy consumed by the terminal to transmit the data dependency between task $i$ and all its predecessors, plus the energy consumed to upload the source code corresponding to that task. This energy can be expressed as:

$$\mathscr{E}_i^{tx} = \sum_{k \in \mathscr{K}} x_i^k \cdot p^{tx} (sr_i^k + \sum_{j \in pred(i)} (x_j^0 \cdot \mathcal{D}_{j,i}^{0,k})) \tag{5.15}$$

where $p^{tx}$ is the radio transmission power [33, 69].

Similarly, $\mathscr{E}_i^{rx}$ is the energy consumed by the terminal to receive the data from remote servers that have processed predecessors tasks of $i$. If the task $i$ is executed locally, then:

$$\mathscr{E}_i^{rx} = \sum_{k \in \mathscr{K}} \sum_{j \in pred(i)} x_j^k \cdot x_i^0 \cdot p^{rx} \cdot \mathcal{D}_{j,i}^{k,0} \tag{5.16}$$

where $p^{rx}$ is the power consumed when the radio interface receive data [33, 68].

Finally, $\mathscr{E}^{idle}$ represents the amount of energy consumed by the network interface when it is in idle mode. This energy depends on the offloading solution and the start times of each offloaded tasks. However, its accurate expression is quite challenging to obtain. To tackle this problem, we consider a pessimistic estimation of $\mathscr{E}^{idle}$, by supposing that all the tasks are transmitted sequentially. Formally, this conservative estimation can be expressed as:

$$\mathscr{E}^{idle} = \max_{k \in \mathscr{K}} \mathscr{E}_k^{idle} \tag{5.17}$$

Here, $\mathscr{E}_k^{idle}$ is the upper bound of the idle energy consumption for the tasks that are performed by the server $k$. This quantity, can be computed as follows:

$$\mathscr{E}_k^{idle} = \sum_{i \in V} p^{idle} \cdot x_i^k \cdot T_i^k + \sum_{i \in V} \sum_{j \in pred(i)} \sum_{m \in \mathscr{K}} p^{idle} \cdot x_i^k \cdot x_j^m \cdot \mathcal{D}_{j,i}^{m,k} \tag{5.18}$$

where $p^{idle}$ is the power consumption when the network interface of the terminal is in idle mode.

## 5.4 Proposed multitasking offloading policy

As introduced earlier, our objective is to decide which task must be offloaded and to which server, in order to minimize the energy consumption of the user terminal. Formally, considering the optimization problem presented in 5.1, we need to determine the best values of $x_i^k$, $\forall i \in V, k \in \mathscr{K}$. Unfortunately, this optimization problem is NP-complete, since it can be easily reduced to a classical scheduling problem by putting the idle energy consumption to zero and removing the constraint $C1$, which is considered as NP-complete [81]. Considering the NP-hardness of the problem, it is difficult to achieve an optimal solution. In order to solve the problem 5.1 and get a feasible solution in a reasonable amount of time, we resort to heuristic based solution. We propose a new

offloading heuristic, named eTOMEC for *Elastic Task graph Offloading in Mobile Edge Computing environment*, which consists of two steps.

The first step of eTOMEC is built on top of the well-known tasks scheduling approach Bottom Level Early Start Time (BL-EST) [82]. The main objective of BL-EST is to map a graph of tasks onto a set of processors, in order to reduce the completion time of the application. It uses an ordered list of tasks to decide with task to schedule first. In this case, they assign to each task a priority, named as bottom level function, and defined as the largest weight path from the current task to the exit task, including the processing time and communication time. Unfortunately, unlike BL-EST, our communication time depends on where the task is scheduled (i.e. offloading server) and thus the necessary delay to send both source code and data to that server. In this case, we propose to assign to each task a new priority considered as an upper-bound of the bottom level function, which corresponds to the worst case offloading decision. Formally, the priority assigned to each task is formulated as following:

$$bl(i) = \max_{k \in \mathscr{K}}(T_i^k + sr_i^k) + \begin{cases} 0 & \text{if succ(i)} = \emptyset \\ \max\limits_{\substack{j \in succ(i) \\ k,m \in \mathscr{K}}} (bl(j) + \mathcal{D}_{i,j}^{k,m}) & \text{otherwise} \end{cases} \tag{5.19}$$

As we can see, from equation 5.19, that the priority assigned to each task is defines as the sum of: (i.) the maximum time necessary to transmit the source code and to schedule the task $i$, among all possible servers $k \in \mathscr{K}$, plus (ii.) the maximum, among all the successors of the task $i$, of the priority assigned to that successor plus the time required to send the data dependency between $i$ and that successor, in the case where the choice of offloading servers is the most unfavorable. This function guarantees an upper-bound of the bottom level function.

Once we assign to each task its priority, we define two sets noted as $S_{ready}$ and $S_{exec}$. $S_{ready}$ is an ordered set according to the priority assigned to each task $i$, i.e. $bl(i)$. At any moment, $S_{ready}$ contains only tasks that are considered as ready to be scheduled, which means tasks that have no predecessor or have all their predecessors already been scheduled. Initially, this list contains only the first task of the DAG, since it has no predecessor. Once scheduled, a task $i$ will be removed and inserted onto the second set $S_{exec}$. Thus, $S_{exec}$ contains only tasks that have been already scheduled. Finally, as in [82], we use heap data structure as implementation for $S_{ready}$ in order to make the

priority-based insertion of tasks more efficient.

As illustrated in Algorithm 4, at each iteration, the task $i$ with the highest priority in $S_{ready}$ is selected to be offloaded on the server that would result to the lowest energy consumption for the mobile terminal. In order to select the best offloading server, we choose the server that minimize the following optimization problem:

$$\begin{aligned} \underset{k\in\mathcal{K}}{\text{Minimize}}\ (&\mathscr{E}_i^l + \mathscr{E}_i^{tx} + \mathscr{E}_i^{rx} + \sum_{j\in pred(i)} \sum_{\substack{m\in\mathscr{K} \\ m\neq 0}} p^{idle}\cdot x_i^k \cdot x_j^m \cdot \mathcal{D}_{j,i}^{m,k} \\ &+ p^{idle}\cdot x_i^k \cdot T_i^k) \\ \text{Subject to:}\quad & C3 \end{aligned} \tag{5.20}$$

where $\mathscr{E}_i^l$, $\mathscr{E}_i^{tx}$ and $\mathscr{E}_i^{rx}$ are defined in 5.14, 5.15 and 5.16, respectively. The fourth term is an upper-bound of the energy consumed in the idle mode, when all the predecessor of the task $i$ have to send their dependency data to the server $k$. The last term correspond the energy consumed in the idle mode when the task $i$ is executed by the server $k$. The basic idea of this objective function is to select the server that minimize the extra amount of energy that could be consumed by the user terminal when scheduling the task $i$ on the server $k$. Finally, the only constraint that we consider in this problem is $C3$, which guarantees that the non-offloadable tasks must be performed at the user's terminal.

After selecting the optimal offloading server $k^*$, we move the task $i$ from the $S_{ready}$ to $S_{exec}$ and we update the decision variable $x_i^k$. We also check if there are new tasks that can be inserted in $S_{ready}$. Finally, we select the next task with the highest priority, and we iterate again until $S_{ready}$ becomes empty.

Once we assign all the tasks, the second step is to ensure that the obtained solution is feasible, which means that it must respect the completion time constraint $C1$. In the case when the obtained solution is not feasible, we propose to merges some tasks in order to reduce the completion time of the application. Basically, we select from the DAG the adjacent tasks $(i, j)$ that are offloaded to different servers and have the highest edge weight. More precisely, the tasks with the most important dependency data transfer time. Thereafter, among the two possible locations, we decide to merge

the two tasks to the server that reduce the most the application competition time. This step is repeated until we obtains a feasible solution.

**Algorithm 4** *Pseudo code of eTOMEC approach*

**Output:** The offloading decisions $\mathcal{X}$, energy consumption $\mathcal{Z}$;
**Input:** Tasks graph G = (V,E), servers and bandwidth allocation;

***Step 1:***
bl ← computePriority(G);
$S_{ready} \leftarrow \emptyset$ ;
insert task 1, entry task , in $S_{ready}$ with key bl(1);
$av^k \leftarrow 0$, $S_{exec} \leftarrow \emptyset$;
**while** ( $S_{ready} \neq \emptyset$) **do**
  i ← extractMax($S_{ready}$);
  compute $ft_i^k$ using equation 5.11, $\forall k \in \mathcal{K}$;
  $k^* \leftarrow$ solution of problem 5.20;
  $x_i^{k^*} \leftarrow 1$, $av^{k^*} \leftarrow ft_i^{k^*}$;
  Insert next ready tasks into $S_{ready}$ using bl as key;
  Add task i to $S_{exec}$
**end while**
***Step 2:***
$S_{merge} \leftarrow$ All edges in G with key as the edge weight;
**while** ($ft_n^0 > t_{max}$ and $S_{merge} \neq \emptyset$ ) **do**
  ft ← $ft_n^0$, (i,j) ← extractMax($S_{merge}$);
  k, m ← servers that performs tasks i and j;
  **if** (k ≠ m) **then**
    $x_i^k = 0$, $x_i^m = 1$, ft1 ← compute new $ft_n^0$;
    $x_i^k = 1$, $x_i^m = 0$, $x_j^m = 0$, $x_j^k = 1$; ft2 ← compute new $ft_n^0$;
    **if** (ft1 > ft2 and ft2 < ft) **then**
      merge task i with j;
    **else if** ft1 < ft **then**
      merge task j with j;
    **end if**
  **end if**
**end while**

## 5.5 Performance evaluation

In this section, we discuss the complexity of the *eTOMEC* offloading policy. Then, we study its performances over different configurations and scenarios.

### 5.5.1 Complexity analysis

In the worst case eTOMEC iterates $n \cdot (K+1)^2$ for the function computePriority using 5.19 to compute priority of tasks. In addition to $n(log(n) + 2(K+1))$ iterations for the step 1, $n$ times in the first while loop. For each iteration, it delete and insert

tasks in the heap which has a complexity of $log(n)$. In addition to $2 \cdot (K+1)$ operations to select the best server for each task, Similarly, it iterates $|E|(log(|E|)+4(K+1))$ in the second while loop, the merge phase. In conclusion, the total complexity of eTOMEC approach is $\mathcal{O}(n \cdot K^2 + n \cdot log(n) + n \cdot K + |E| \cdot log(|E|) + |E| \cdot K)$. That makes eTOMEC a quick algorithm to solve problem 5.1.

### 5.5.2 Results analysis

In this section, we discuss the simulation results of eTOMEC. We consider a MEC environment with a wireless access network composed of one AP and 4 edge servers and 1 remote cloud. In addition, we consider that the edge servers have the same computing capability to perform the tasks. As in [68, 69], we assume that both the cloud and edge servers has a computing capability of 5 *Giga cycles/s*. Moreover, we suppose that user terminal has a local computing capacity of 1 *Giga cycles/s*.

**Table 5.1:** *The parameter setting of the network interface*

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| B | 5 Mhz | $\upsilon$ | 50 |
| $\mathcal{B}$ | 256 | $b^{rx} = b^{tx}$ | $10^{-3}$ |
| $\sigma^2$ | $5 \times 10^{-5}$ | $p^{tx}$ | 1.28 Watts |
| $p^{rx}$ | 1.18 Watts | $p^{idle}$ | 0.1 Watts |
| $p_{ap}$ | 10 Watts | $h^{tx}$ | 0.1 |
| $d$ | 10 | $h^{rx}$ | 0.1 |

Table 5.1 shows the wireless network parameter settings as presented in [37, 42, 43]. For the backhaul network, we use the parameters presented in [68, 69]. Finally, the user wants to perform one application composed of 14 tasks as shown in Figure 5.2. To understand the application graph effect, we assume that all the tasks require the same computing resource and transmit the same amount of data. Every task requires 30 *Giga cycles* [33, 68]. The dependencies between tasks is set to $300KB$ and the source code data to $2MB$. The results of eTOMEC are compared with the following approaches:

- **WFM** [83]: A list-based scheduling approach. First, it determines the level of the tasks in a breadth-first-search traversal of the DAG. Then, the tasks in each level are assigned to different servers using a round robin algorithm.
- **HEM** [83]: A clustering-based approach. Initially, it assigns the (K+2) heaviest tasks to the servers. Then, it sorts the edges in decreasing order of their weight.

The endpoints, tasks, of each edge are clustered if the completion time does not increase.

- **MCOP** [33, 84]: A DAG-based offloading algorithm with one single edge server.

### Performance comparison

In Figures 5.4 and 5.5, we investigate the offloading performance of the four offloading policies over different DAG-based applications. We notice that eTOMEC outperforms WFM, HEM and MCOP in both of total energy consumption and completion time. The only exception is for the sequential graph, this is because of the consideration of offloading to multi-edge servers by eTOMEC, in the sequential graph all the offloaded tasks are assigned to the same edge server in order to reduce the communication cost between those tasks. In addition, the figures show that depending on the application graph eTOMEC achieves better performance than the classical tasks scheduling approaches WFM and HEM. For example, eTOMEC and WFM are 6.94% better than HEM in energy consumption and 25.67% in completion time with parallel graph. However, eTOMEC and HEM have the same performance for Fan In/Fan Out. Furthermore, the energy consumption of the application depends highly on the graph. This conclusion means that the composition of the application and the tasks-dependeny graph affect the energy consumption and the completion time of the application.

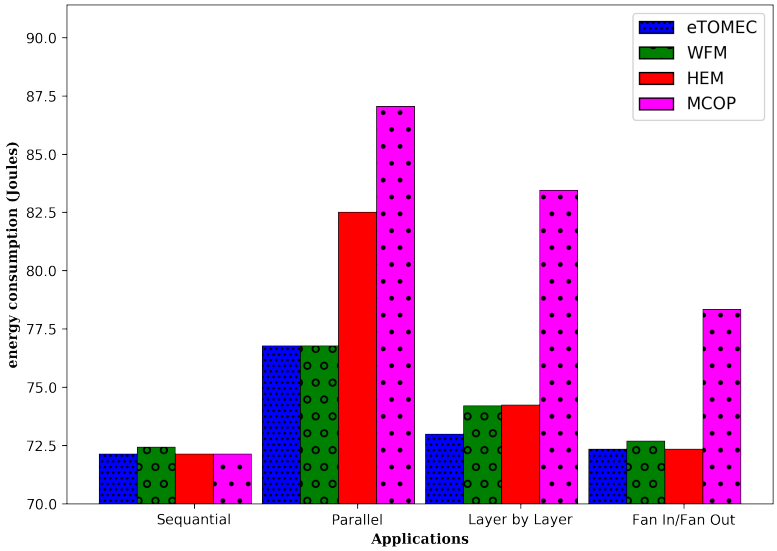



**Figure 5.4:** *Energy consumption comparison for different application graphs*

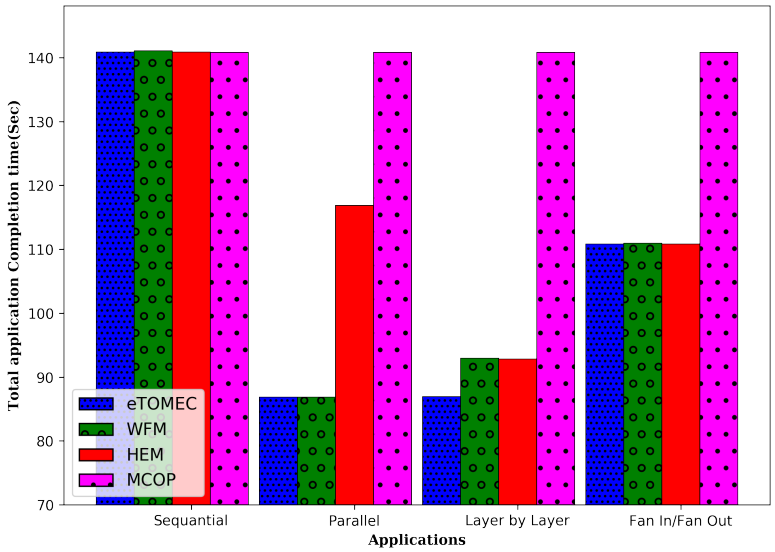



**Figure 5.5:** *Completion time comparison for different application graphs*

### The effect of the number of the available edge servers

In order to evaluate the effect of the application graph on the performance of the computation offloading, we study in Figures 5.6 and 5.7, the total energy consumption the completion time when varying the number of available edge servers, when using our approach (eTOMEC). We observe that energy consumption is highly related to the degree of tasks that are executed in the user terminal, which is the case of tasks 1 and 14. For instance, in the case of sequential graph the user terminal uploads the data related to the dependency between tasks 1 and 2 and download the data between tasks 13 to the task 14. On the other hand, in the case of parallel graph the user terminal have to send and receive data from several tasks. Hence, the additional energy consumption is due to upload and the download of data from remote and local tasks. Moreover, we note that completion time decreases when we increase the number of available edge servers, except for sequential graph. In addition, we can observe that the completion time reaches a constant value after a given number of available edge servers. This number depends on the graph, for example, for sequential graph it is 1, but it is equal to 12, 4 and 2 for parallel, Layer by Layer and Fan In/Fan Out graphs, respectively. This is due to the maximum number of tasks that we can perform in parallel. In Figure 5.6 depicts also the same behavior for the total energy consumption. As conclusion, the application graph affects strongly the performance of offloading.

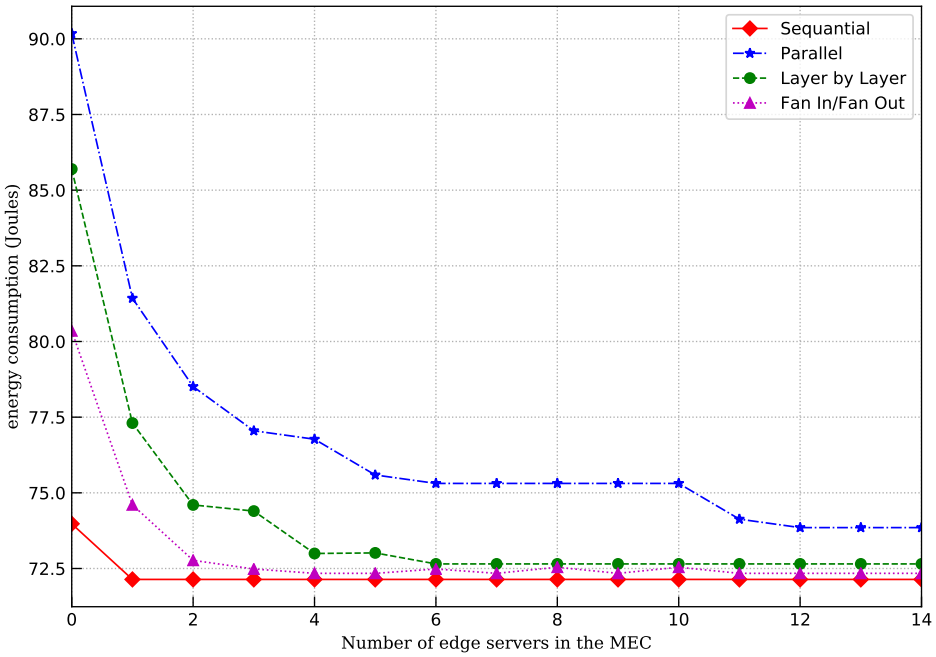



**Figure 5.6:** *The energy consumption of eTOMEC for different application graphs*

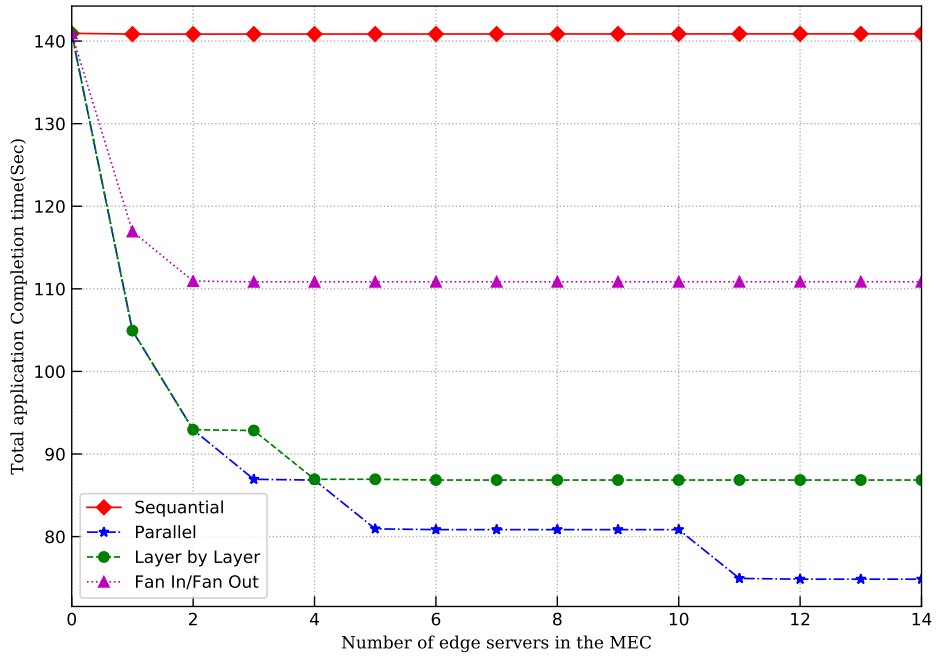



**Figure 5.7:** *Completion time of eTOMEC for different application graphs*

### The effect of the task-dependency graph

To analyze more deeply the effect of the application's graph on the performance of the offloading approach, we evaluate the performance of the different approach on a generic application. To generate a DAG for this generic application, we use Erdős–Rényi graph generation model [85]. This model constructs a DAG composed of $n$ tasks by connecting the tasks between themselves using a dependency probability $p$. In order word, the dependency between every two tasks is the set to a given probability $p$. To have a representative sample, we generate 200 DAGs for each number of tasks $n$ and dependency probability $p$. The Figures bellow depict the performance of different approach for the generated task-dependency graphs.

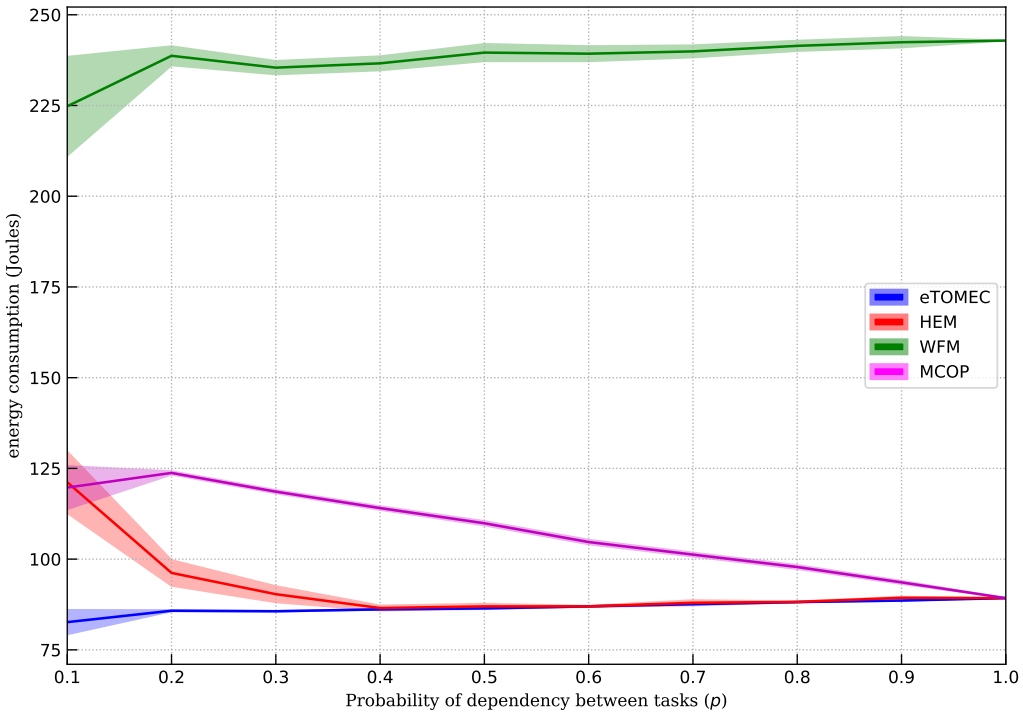



**Figure 5.8:** *Average energy consumption under probability p for DAG of 50 tasks.*

In Figure 5.8 and 5.9, we study the average total energy consumption and completion time of cloud-vision composed of 50 tasks according to different probability $p$. We observe that when the dependency between tasks is low eTOMEC achieves better performance. For example, for $p = 0.1$ the energy consumption is 82.63 joules, 121.18 joules and 119.72 joules for eTOMEC, HEM and MCOP, respectively, which makes eTOMEC 30% better than the closest approach (MCOP). However, when the probability $p$ increase eTOMEC, HEM and MCOP become equivalents. This is due to fact that

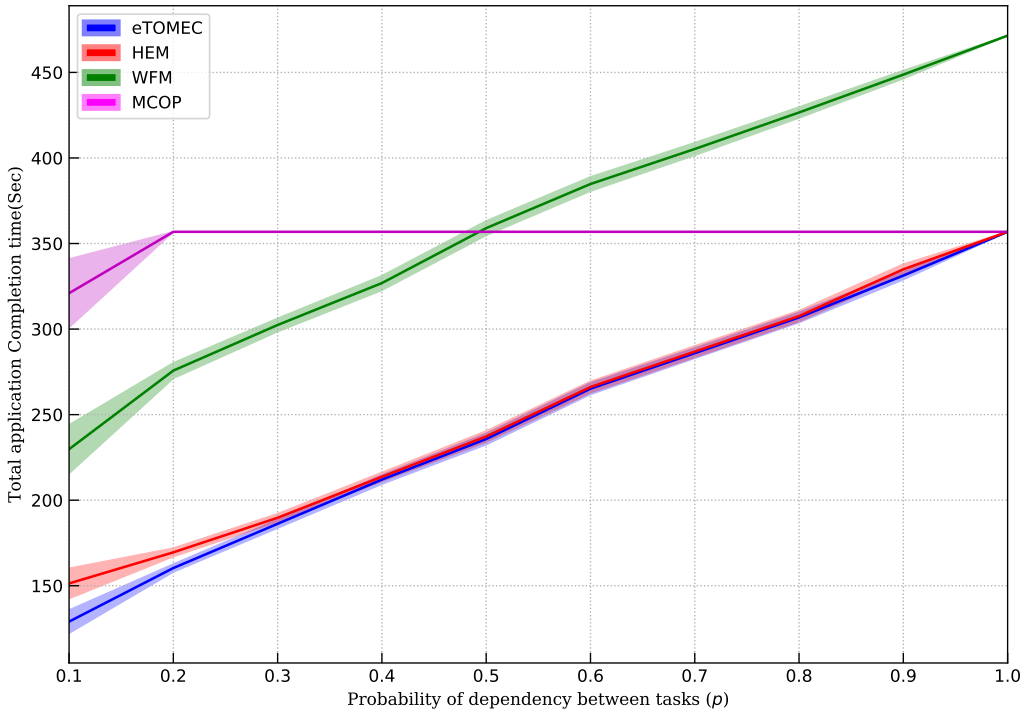



**Figure 5.9:** *Average completion Time under probability p for DAG of 50 tasks.*

when $p$ increases the graph has more edges. Consequently, the offloading approach uses less edge servers to performance the tasks to reduce the energy consumption and the completion time.

In Figure 5.10 and 5.11, we investigate the effect of number of tasks in the application graph on the offloading approach where the probability $p = 0.2$. From the figure, we note that when the number of tasks is less than 20, the approaches eTOMEC, HEM and WFM have the similar performances and they are better than MCOP. However, when the number of tasks goes up eTOMEC and HEM become better. Our proposal eTOMEC achieves better performance in both energy consumption and completion time than HEM. As a conclusion from these figure, eTOMEC offloading approach is very efficient for multitasking application in multi-edge server MEC environment.

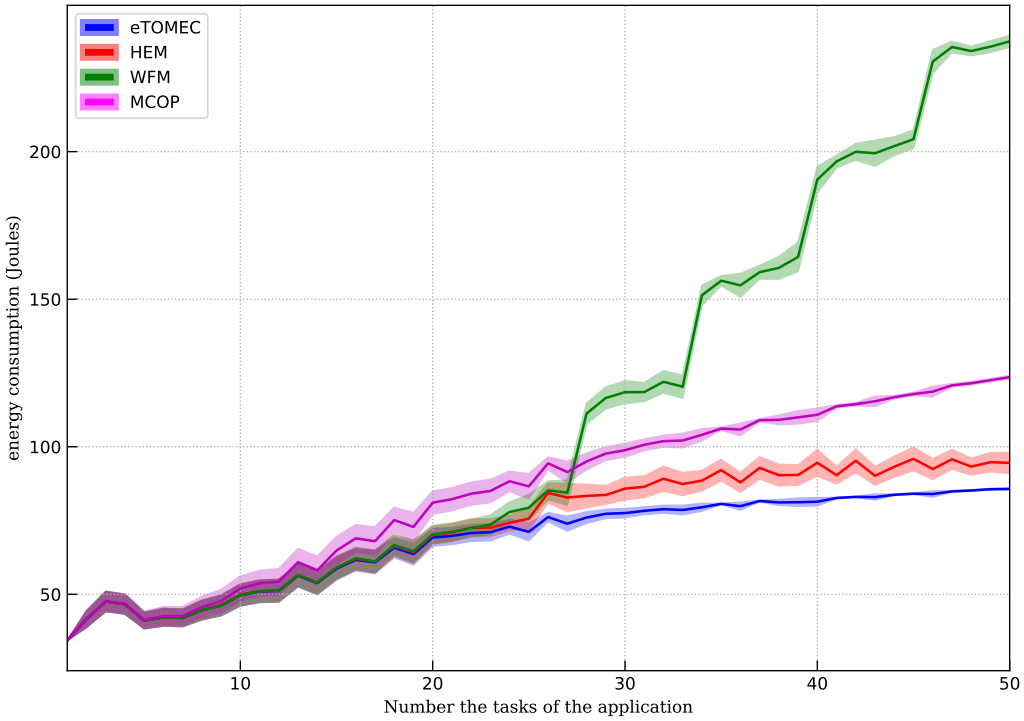



**Figure 5.10:** *Average energy consumption under number of tasks in the graph ($p = 0.2$).*

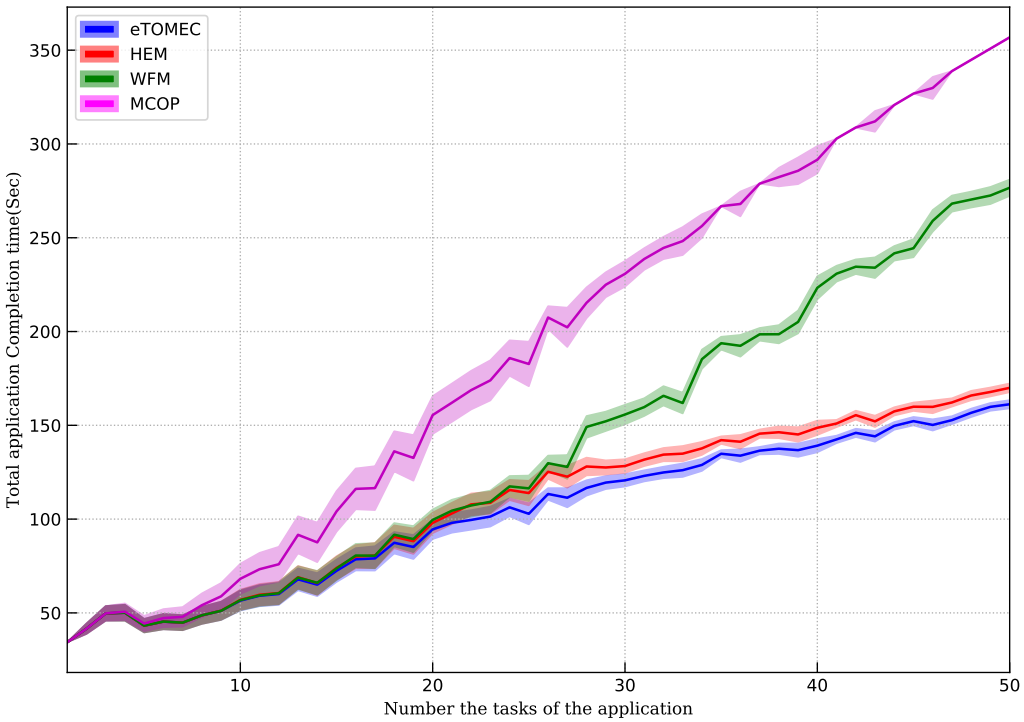



**Figure 5.11:** *Average completion time under number of tasks in the graph ($p = 0.2$).*

## 5.6 Conclusion

This chapter presents a new computation offloading approach for multitasking application in multi-edge server MEC environment. The proposed offloading approach decides which tasks should be offload and to which server. We formalize this offloading problem as a Zero-one Integer Programming problem. Thereafter, and due to the NP-hardness of the problem, we propose a heuristic approach named eTOMEC. Using our efficient offloading algorithm, we where able to scale the number of concurrent offloaded tasks accordingly with the number of available edge servers, while taking into consideration tasks dependencies.

The numerical results show that our proposed offloading approach, eTOMEC, achieves better performances in both of completion time and energy consumption than several literature approaches. In addition, we show the efficiency of eTOMEC for multitasking application whatever the task-dependency graph. We conclude that the performance of computation offloading approach relies strongly on the application's task-dependency graph.

CHAPTER 6

# Conclusion and future work

This chapter summaries the research works on computation offloading efficiency for multi-edge server MEC presented in this manuscript. In addition, it highlights some directions to pursue in future works in this area.

## 6.1 Summary of contributions

The new generation of mobile terminals, such as smartphones, tablets, and ultra-books, suffers from lacking computing resource and battery autonomy. The computation offloading technique is a promising solution to overcome this problem since it promises to save battery power and enriches the computing capabilities of the mobile terminals. In this thesis, we have explored the Mobile Edge Computing environment as a way to extend the computing capabilities of the mobile terminals through the offloading. In this thesis, we introduce new offloading approaches to improve the performance of a resource-hungry application in MEC.

Chapter 3 investigates the offloading computation for multi-user multi-edge server scenario. It proposes an efficient offloading policy, which considers the offloading decision and the edge server selection for mobile terminals. The proposed policy distinguish between : 1.) applications that are split-up into client-server model where the server is already available on the MEC and where the application need to offload only the input data and 2.) the applications that need to transmit its source code in addition to the input data to the MEC. Based on the application model, the proposed approach

determines the offloading decision and the best target edge server where to perform computation in order to minimize the terminals' energy consumption.

Chapter 4 extends the offloading policy proposed in chapter 3 to two-tired MEC environment. It studied the impact of the remote cloud on the performance of the computation offloading. The new approach considers all the servers, including the cloud, and offloads to the best ones in terms of energy consumption. To assess the behavior of this approach on real scenario, we designed and implemented a new computation offloading middleware for Android Applications that uses the proposed algorithm for the computation offloading. Using this middleware, we studied the performances of different real applications.

Finally, chapter 5 studies the effect of the application on the offloading performance. Basically, it considers multitasking application where tasks are interdependent on each other according to a directed graph. The basic idea is to use multiple edge server in order to maximize the number of tasks that can be executed in parallel and thus maximizing the offloading performance such as the energy consumption and the completion time of the application.

## 6.2 General discussion

At the end of this thesis, we conclude that the computation offloading to MEC is a very promising technique to enhance the mobile terminal capabilities. However, its efficiency depends on many considerations. The application's characteristics play an important role to determine whether if the offloading is beneficial or not and how it affects the offloading location in the MEC. For example, the applications that send a huge amount of data compared to the required computation are preferred to be offloaded to an edge server near to the user. However, applications that required more computation can be offloaded either to MEC servers or to the remote cloud. Moreover, we note that determining the metric to measure to performance depends also on the application's requirements. Minimizing the energy consumption gives a good solution, but for some applications it is more important to consider the completion time. A hybrid metric can be a good option. Finally, we also noticed that the way an application is developed impacts considerably the offloading. Indeed, the architecture of the application and its task-dependency graph determines how the offloading can be beneficial.

## 6.3 Future works and perspectives

Even through that this thesis proposes efficient policies, we believe that they can be further extended to more complex cases directions to improve and enhance these policies. In the following, we list some directions that can be investigated in future works:

- **`Network bandwidth allocation:`** This thesis was mainly focused on the offloading decision and the selection of the edge server to perform offloaded tasks, the communication resource, network bandwidth, was assumed to be allocated in advance. In practice, the wireless bandwidth affects highly the offloading. Consequently, it is very important to extend the proposed approaches by considering wireless bandwidth allocation to each mobile terminal. The resulting joint-optimization problem will be more complex to solve, new approaches are required to solve it.

- **`Computation resource allocation:`** Similar to the bandwidth allocation, the computation resource allocation is supposed to be done in advance. However, the efficiency of the offloading depends strongly on the processing time which depends on this resource. So, it is important to consider dynamic computation resource allocation and study the effect of this allocation on the performances of the applications. As for the bandwidth allocation, the resulting offloading problem will be a joint optimization problem.

- **`Offloading middleware:`** The experiments presented in this thesis considered only the mono-task applications. The proposed offloading middleware do not consider multitasking applications. However, it is important to evaluate, using real experimentation, the performance of multitasking offloading applications. In addition, it is important to consider different virtualization technologies in order to evaluate the best way to implement a MEC platform. Container-based offloading could be a promising solution for the middleware implementation.

In addition to the directions listed above, we think that the following general directions must also be addressed as long term future works on computation offloading in MEC:

- **`User mobility:`** One of the characteristics of mobile terminals is the mobility of the user. Indeed, in such an environment the user moves from location to another

and moving from one base station to another. This mobility introduces a new challenge to the computation offloading. Indeed, the geographical proximity of the edge server has a significant impact on the offloading performances. One possible approach is to select the edge server according not only on the current location of the mobile terminal but also to the future locations.

- **`Pricing:`** It is clear that the MEC's servers are deployed by a third party. Thus, a pricing model for the computation plays a crucial role. The user must know how much he pays when it offloads its computation to the MEC or to the cloud in order to decide where to offload.

- **`Security:`** Computation offloading means that the user sends the source code and the data of its applications to a third party, which can result in a big security issue. Indeed, How to be sure that no copy of the user data is made? In addition, the edge server need also to trust the user source code. Indeed, how to be sure that the user is not transmitting a virus code to the MEC? These questions are very challenging in the context of computing offloading. However, few works in the literature address these issues. Consequently, it is very important to consider the security as an important issue for real MEC deployments.

APPENDIX A

# Appendix

## A.1 The Proof of Theorem 3.2 of chapter 3

Given the objective function of problem 3.24, which is a function with the variable $a_{m,n}$, we can find the minimum following the derivative sign rules [67]. Let $\mathcal{F}_{m,n}$ be the derivative of the objective function, $\mathcal{Z}^{e,k}_{m,n} + \mathcal{Z}^{l}_{m,n}$, to the variable $a_{m,n}$. $\mathcal{F}_{m,n}$ is given by the following expression:

$$\mathcal{F}_{m,n} = up_{m,n} \cdot \left(\frac{P^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}}{w^{tx}_{m,n}} + \frac{\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}}{W^{bh,k}_{m}}\right) + \gamma_{m,n} \cdot \left(\frac{\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}}{c^k} - \beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^2 - \frac{1 - \beta_{m,n}}{f_{m,n}}\right) \quad \text{(A.1)}$$

By unification of the denominators, the equation A.1 can be rewritten as follows:

$$\mathcal{F}_{m,n} = up_{m,n} \cdot \left(\frac{[P^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}] \cdot W^{bh,k}_{m} + [\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot w^{tx}_{m,n}}{w^{tx}_{m,n} \cdot W^{bh,k}_{m}}\right) + \gamma_{m,n} \cdot \left(\frac{[\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot f_{m,n} - [\beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^3 \cdot c^k] - [1 - \beta_{m,n}] \cdot c^k}{c^k \cdot f_{m,n}}\right)$$

The derivative function $\mathcal{F}_{m,n}$ is a constant and does not change when the variable $a_{m,n}$ does. Consequently, we have three cases according to the derivative sign rules [67]:

**Case 1:** $\mathcal{F}_{m,n} < 0$, in this case the objective function of problem 3.24 is monotonically decreasing. As a result, its minimum is achieved at $a_{m,n} = 1$. This case occurs when:

$$\frac{up_{m,n}}{\gamma_{m,n}} < \frac{w^{tx}_{m,n} \cdot W^{bh,k}_{m}}{c^k \cdot f_{m,n}} \times \frac{([\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot f_{m,n} - [\beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^3 \cdot c^k] - [1 - \beta_{m,n}] \cdot c^k)}{([P^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}] \cdot W^{bh,k}_{m} + [\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot w^{tx}_{m,n})}$$

**Case 2:** $\mathcal{F}_{m,n} > 0$, there the objective function of problem 3.24 is monotonically increasing. Consequently, its minimum is achieved at $a_{m,n} = 0$. This case occurs when:

$$\frac{up_{m,n}}{\gamma_{m,n}} > \frac{w^{tx}_{m,n} \cdot W^{bh,k}_{m}}{c^k \cdot f_{m,n}} \times \frac{([\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot f_{m,n} - [\beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^3 \cdot c^k] - [1 - \beta_{m,n}] \cdot c^k)}{([P^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}] \cdot W^{bh,k}_{m} + [\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot w^{tx}_{m,n})}$$

**Case 3:** $\mathcal{F}_{m,n} = 0$, there the objective function of problem 3.24 is constant and does not change. So, at the values of $a_{m,n}$ give the same cost. This case occurs when:

$$\frac{up_{m,n}}{\gamma_{m,n}} = \frac{w^{tx}_{m,n} \cdot W^{bh,k}_{m}}{c^k \cdot f_{m,n}} \times \frac{([\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot f_{m,n} - [\beta_{m,n} \cdot \kappa_{m,n} \cdot (f_{m,n})^3 \cdot c^k] - [1 - \beta_{m,n}] \cdot c^k)}{([P^{tx}_{m,n} \cdot \beta_{m,n} + 1 - \beta_{m,n}] \cdot W^{bh,k}_{m} + [\beta_{m,n} \cdot P^{idle}_{m,n} + 1 - \beta_{m,n}] \cdot w^{tx}_{m,n})}$$

Ending of the proof. ■

# Publications

Based on the research work presented in this thesis, some papers have been published in international conferences and journals:

i. **Conferences:**

1. Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Elastic Offloading of Multitasking Applications to Mobile Edge Computing". In: *Proceedings of the 22st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2019, Miami Beach, FL, USA, November 25 - November 29, 2019*. 2019, pp. 1–8. DOI: 10.1145/3345768.3355926
2. Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud". In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2018, Montreal, QC, Canada, October 28 - November 02, 2018*. 2018, pp. 137–145. DOI: 10.1145/3242102.3242133

ii. **Journals:**

1. Houssemeddine Mazouzi, Khaled Boussetta, and Nadjib Achir. "Maximizing mobiles energy saving through tasks optimal offloading placement in two-tier cloud: A theoretical and an experimental study". In: *Computer Communications* 144 (2019), pp. 132–148. DOI: 10.1016/j.comcom.2019.05.017
2. Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "DM2-ECOP: An Efficient Computation Offloading Policy for Multi-user Multi-cloudlet Mobile Edge Computing Environment". In: *ACM Transactions on Internet Technology*. 19.2 (2019), 24:1–24:24. DOI: 10.1145/3241666

# Bibliography


[1] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction". In: *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 64–76 *Cited on page 2.*

[2] Mazliza Othman and Stephen Hailes. "Power conservation strategy for mobile computers using load sharing". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 2.1 (1998), pp. 44–51 *Cited on page 3.*

[3] Alexey Rudenko et al. "Saving portable computer battery power through remote process execution". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 2.1 (1998), pp. 19–26 *Cited on page 3.*

[4] Grace Alexandra Lewis. "Software Architecture Strategies for Cyber-Foraging Systems". In: *Dissertation Carnegie Mellon University in Pittsburgh* (2016) *Cited on pages 3, 4, 6, 10, 29.*

[5] Rajesh Krishna Balan et al. "Tactics-based remote execution for mobile computing". In: *Proceedings of the 1st international conference on Mobile systems, applications and services*. ACM. 2003, pp. 273–286 *Cited on page 3.*

[6] Mahadev Satyanarayanan et al. "The role of cloudlets in hostile environments". In: *IEEE Pervasive Computing* 12.4 (2013), pp. 40–49 *Cited on pages 3, 26.*

[7] Debessay Fesehaye et al. "Impact of cloudlets on interactive mobile cloud applications". In: *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*. IEEE. 2012, pp. 123–132 *Cited on pages 3, 26, 37.*

[8] Pavel Mach and Zdenek Becvar. “Mobile edge computing: A survey on architecture and computation offloading”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1628–1656 *Cited on pages 4, 6, 10.*

[9] Mahadev Satyanarayanan. “Edge computing for situational awareness”. In: *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE. 2017, pp. 1–6 *Cited on page 9.*

[10] Mahadev Satyanarayanan. “The emergence of edge computing”. In: *Computer* 50.1 (2017), pp. 30–39 *Cited on page 9.*

[11] Noriyuki Takahashi, Hiroyuki Tanaka, and Ryutaro Kawamura. “Analysis of process assignment in multi-tier mobile cloud computing and application to edge accelerated web browsing”. In: *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE. 2015, pp. 233–234 *Cited on page 10.*

[12] Yingjuan Shi et al. “The fog computing service for healthcare”. In: *2015 2nd International Symposium on Future Information and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech)*. IEEE. 2015, pp. 1–5 *Cited on page 10.*

[13] Meng Li, Pengbo Si, and Yanhua Zhang. “Delay-tolerant data traffic to software-defined vehicular networks with mobile edge computing in smart city”. In: *IEEE Transactions on Vehicular Technology* 67.10 (2018), pp. 9073–9086 *Cited on page 10.*

[14] Oleksandr Zhdanenko et al. “Demonstration of Mobile Edge Cloud for 5G Connected Cars”. In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2019, pp. 1–2 *Cited on page 10.*

[15] Linh Van Ma et al. “Nfv-based mobile edge computing for lowering latency of 4k video streaming”. In: *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. 2018, pp. 670–673 *Cited on page 10.*

[16] Pavel Mach and Zdenek Becvar. “Mobile edge computing: A survey on architecture and computation offloading”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1628–1656 *Cited on page 10.*

[17] ESTI GROUP SPECIFICATION. *Multi-access Edge Computing (MEC) Phase 2: Use Cases and Requirements*. 2017. URL: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/02.01.01_60/gs_MEC002v020101p.pdf (visited on 09/03/2019) *Cited on page 11.*

[18] Mike Jia, Jiannong Cao, and Weifa Liang. "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks". In: *IEEE Transactions on Cloud Computing* PP.99 (2015) *Cited on pages 15, 29, 45, 47, 48, 66.*

[19] Zichuan Xu et al. "Efficient Algorithms for Capacitated Cloudlet Placements". In: *IEEE Transactions on Parallel and Distributed Systems* 27.10 (2016), pp. 2866–2880 *Cited on pages 15, 28, 29, 45, 47, 48, 66.*

[20] Hong Yao et al. "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing". In: *Concurrency and Computation: Practice and Experience* 29.16 (2017) *Cited on pages 15, 26.*

[21] Longjie Ma, Jigang Wu, and Long Chen. "DOTA: Delay Bounded Optimal Cloudlet Deployment and User Association in WMANs". In: *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press. 2017, pp. 196–203 *Cited on pages 15, 26, 45, 47, 66.*

[22] Hyun-Suk Lee and Jang-Won Lee. "Task offloading in heterogeneous mobile cloud computing: Modeling, analysis, and cloudlet deployment". In: *IEEE Access* 6 (2018), pp. 14908–14925. DOI: 10.1109/ACCESS.2018.2812144 *Cited on page 15.*

[23] Yuanzhe Li and Shangguang Wang. "An energy-aware edge server placement algorithm in mobile edge computing". In: *2018 IEEE International Conference on Edge Computing (EDGE)*. IEEE. 2018, pp. 66–73. DOI: 10.1109/EDGE.2018.00016 *Cited on page 16.*

[24] Anwesha Mukherjee, Debashis De, and Deepsubhra Guha Roy. "A power And latency aware cloudlet selection strategy for multi-cloudlet environment". In: *IEEE Transactions on Cloud Computing* PP.99 (2016), pp. 1–14 *Cited on page 16.*

[25] Deepsubhra Guha Roy et al. "Application-aware cloudlet selection for computation offloading in multi-cloudlet environment". In: *The Journal of Supercomputing* (2016), pp. 1–19 *Cited on pages 16, 26.*

[26] Mike Jia et al. “Cloudlet load balancing in wireless metropolitan area networks”. In: *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. Vol. 2016-July. 2016, pp. 1–9 *Cited on pages 16, 26, 66.*

[27] Qiliang Zhu et al. “Task offloading decision in fog computing system”. In: *China Communications* 14.11 (2017), pp. 59–68 *Cited on page 16.*

[28] Arash Bozorgchenani, Daniele Tarchi, and Giovanni Emanuele Corazza. “An Energy and Delay-Efficient Partial Offloading Technique for Fog Computing Architectures”. In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6 *Cited on page 16.*

[29] Xile Shen et al. “Edge Computing Sever Selection in Fog Radio Access Networks”. In: *2018 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE. 2018, pp. 287–291. DOI: 10.1109/ICCChinaW.2018.8674471 *Cited on page 16.*

[30] Tao Ouyang et al. “Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach”. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE. 2019, pp. 1468–1476. DOI: 10.1109/INFOCOM.2019.8737560 *Cited on page 16.*

[31] Meng-Hsi Chen, Ben Liang, and Min Dong. “Joint offloading decision and resource allocation for multi-user multi-task mobile cloud”. In: *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, pp. 1–6 *Cited on pages 17, 26, 27, 48, 66, 101.*

[32] Xu Chen et al. “Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing”. In: *IEEE/ACM Transactions on Networking* 24.5 (2016), pp. 2795–2808 *Cited on pages 17, 26–28, 37, 47, 48, 63, 66.*

[33] Songtao Guo et al. “Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing”. In: *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. Vol. 2016-July. IEEE. 2016, pp. 1–9 *Cited on pages 17, 19, 20, 26–28, 47, 63, 98, 100, 106–108, 112, 113.*

[34] Keke Gai, Meikang Qiu, and Hui Zhao. “Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing”. In: *Journal of Parallel and Distributed Computing* 111 (2018), pp. 126–135 *Cited on page 17.*

[35] Yuyi Mao et al. “Power-delay tradeoff in multi-user mobile-edge computing systems”. In: *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE. 2016, pp. 1–6 *Cited on pages 17, 29.*

[36] Feng Wang, Jie Xu, and Zhiguo Ding. “Multi-antenna NOMA for computation offloading in multiuser mobile edge computing systems”. In: *IEEE Transactions on Communications* 67.3 (2018), pp. 2450–2463. DOI: 10.1109/TCOMM.2018.2881725 *Cited on page 17.*

[37] Tuyen X Tran and Dario Pompili. “Joint task offloading and resource allocation for multi-server mobile-edge computing networks”. In: *IEEE Transactions on Vehicular Technology* 68.1 (2019), pp. 856–868 *Cited on pages 18, 48, 107, 112.*

[38] Yucen Nan et al. “Adaptive Energy-Aware Computation Offloading for Cloud of Things Systems”. In: *IEEE Access* 5 (2017), pp. 23947–23957 *Cited on page 18.*

[39] Yucen Nan et al. “A dynamic tradeoff data processing framework for delay-sensitive applications in Cloud of Things systems”. In: *Journal of Parallel and Distributed Computing* 112 (2018), pp. 53–66 *Cited on page 18.*

[40] Chongyu Zhou, Chen-Khong Tham, and Mehul Motani. “Online Auction for Truthful Stochastic Offloading in Mobile Cloud Computing”. In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–6 *Cited on page 18.*

[41] Dong Huang, Ping Wang, and Dusit Niyato. “A dynamic offloading algorithm for mobile computing”. In: *IEEE Transactions on Wireless Communications* 11.6 (2012), pp. 1991–1995 *Cited on pages 18, 19.*

[42] Thinh Quang Dinh et al. “Offloading in mobile edge computing: Task allocation and computational frequency scaling”. In: *IEEE Transactions on Communications* 65.8 (2017), pp. 3571–3584 *Cited on pages 18, 48, 112.*

[43] Weiwei Chen, Dong Wang, and Keqin Li. “Multi-user multi-task computation offloading in green mobile edge cloud computing”. In: *IEEE Transactions on Services Computing* (2018) *Cited on pages 18, 48, 106, 112.*

[44] Manuel Osvaldo J Olguin Munoz et al. “EdgeDroid: An Experimental Approach to Benchmarking Human-in-the-Loop Applications”. In: *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM. 2019, pp. 93–98. DOI: 10.1145/3301293.3302353 *Cited on page 19.*

[45] Bowen Zhou. “Code Offloading and Resource Management Algorithms for Heterogeneous Mobile Clouds”. PhD thesis. 2018 *Cited on page 19.*

[46] Jose I Benedetto et al. “MobiCOP: A Scalable and Reliable Mobile Code Offloading Solution”. In: *Wireless Communications and Mobile Computing* 2018 (2018) *Cited on pages 19, 22, 73.*

[47] S. Eman Mahmoodi, R.N. Uma, and K.P. Subbalakshmi. “Optimal Joint Scheduling and Cloud Offloading for Mobile Applications”. In: *IEEE Transactions on Cloud Computing* 7161.c (2016), pp. 1–1 *Cited on pages 19, 100, 101.*

[48] Shiqiang Wang, Murtaza Zafer, and Kin K Leung. “Online placement of multi-component applications in edge computing environments”. In: *IEEE Access* 5 (2017), pp. 2514–2533. DOI: 10.1109/ACCESS.2017.2665971 *Cited on page 19.*

[49] Lei Yang et al. “Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds”. In: *IEEE Transactions on Services Computing* (2019) *Cited on pages 20, 98, 100.*

[50] Vincenzo De Maio and Ivona Brandic. “First hop mobile offloading of dag computations”. In: *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press. 2018, pp. 83–92. DOI: 10.1109/CCGRID.2018.00023 *Cited on page 20.*

[51] M Yusuf Özkaya et al. “A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning”. In: *IPDPS 2019-33rd IEEE International Parallel & Distributed Processing Symposium*. IEEE. 2019, pp. 1–11 *Cited on page 20.*

[52] Vinay Kumar, CP Katti, and PC Saxena. “A Novel Task Scheduling Algorithm for Heterogeneous Computing”. In: *International Journal of Computer Applications* 85.18 (2014) *Cited on page 20.*

[53] Eduardo Cuervo et al. “MAUI: making smartphones last longer with code offload”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010*. ACM, pp. 49–62 *Cited on pages 22, 37, 73.*

[54] Robert C Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002 *Cited on page 22.*

[55] Byung-Gon Chun et al. “Clonecloud: elastic execution between mobile device and cloud”. In: *European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, Salzburg, Austria, April 10-13, 2011*. ACM, pp. 301–314 *Cited on pages 22, 73.*

[56] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. “DM2-ECOP: An Efficient Computation Offloading Policy for Multi-user Multi-cloudlet Mobile Edge Computing Environment”. In: *ACM Transactions on Internet Technology*. 19.2 (2019), 24:1–24:24. DOI: 10.1145/3241666 *Cited on pages 25, 127.*

[57] Ying Gao et al. “Are cloudlets necessary?” In: *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139* (2015) *Cited on pages 29, 47, 77.*

[58] Doyub Kim et al. “Near-exhaustive precomputation of secondary cloth effects”. In: *ACM Trans. Graph.* 32.4 (July 2013), 87:1–87:8 *Cited on page 30.*

[59] Song Wu et al. “Container-based cloud platform for mobile computation offloading”. In: *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE. 2017, pp. 123–132 *Cited on page 30.*

[60] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005. DOI: 10.1017/CBO9780511841224 *Cited on pages 34, 35, 103, 104.*

[61] Aaron Carroll, Gernot Heiser, et al. “An Analysis of Power Consumption in a Smartphone.” In: *USENIX annual technical conference*. Vol. 14. Boston, MA. 2010, pp. 21–21 *Cited on pages 36, 48, 107.*

[62] Antti P Miettinen and Jukka K Nurminen. “Energy Efficiency of Mobile Clients in Cloud Computing.” In: *HotCloud* 10 (2010), pp. 4–4 *Cited on page 36.*

[63] Swetank Kumar Saha et al. “Revisiting 802.11 power consumption modeling in smartphones”. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A*. IEEE. 2016, pp. 1–10 *Cited on page 36.*

[64] Sven O Krumke and Clemens Thielen. “The generalized assignment problem with minimum quantities”. In: *European Journal of Operational Research* 228.1 (2013), pp. 46–55 *Cited on page 39.*

[65] Jiafu Tang et al. “Using lagrangian relaxation decomposition with heuristic to integrate the decisions of cell formation and parts scheduling considering intercell moves”. In: *IEEE Transactions on Automation Science and Engineering* 11.4 (2014), pp. 1110–1121 *Cited on pages 39, 46, 47.*

[66] Marshall L Fisher. “The Lagrangian relaxation method for solving integer programming problems”. In: *Management science* 50.12_supplement (2004), pp. 1861–1871 *Cited on pages 39, 45–47.*

[67] Mark Ryan. *Calculus workbook for Dummies*. Wiley Publishing, Inc., 2005. ISBN: 076458782X,9780764587825 *Cited on pages 44, 125.*

[68] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. “Dm2-ecop: An efficient computation offloading policy for multi-user multi-cloudlet mobile edge computing environment”. In: *ACM Transactions on Internet Technology (TOIT)* 19.2 (2019), p. 24 *Cited on pages 48, 98, 100, 106–108, 112.*

[69] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. “Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud”. In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM. 2018, pp. 137–145 *Cited on pages 48, 98, 101, 107, 112.*

[70] Jessica Oueis et al. “On the impact of backhaul network on distributed cloud computing”. In: *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*. IEEE. 2014, pp. 12–17 *Cited on page 48.*

[71] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. “Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud”. In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2018, Montreal,*

*QC, Canada, October 28 - November 02, 2018*. 2018, pp. 137–145. DOI: 10.1145/3242102.3242133 *Cited on pages 61, 62, 127.*

[72] Houssemeddine Mazouzi, Khaled Boussetta, and Nadjib Achir. "Maximizing mobiles energy saving through tasks optimal offloading placement in two-tier cloud: A theoretical and an experimental study". In: *Computer Communications* 144 (2019), pp. 132–148. DOI: 10.1016/j.comcom.2019.05.017 *Cited on pages 61, 127.*

[73] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. "Performance evaluation of single-tier and two-tier cloudlet assisted applications". In: *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE. 2017, pp. 302–307 *Cited on page 62.*

[74] Abbas Kiani et al. "A Two-Tier Edge Computing Based Model for Advanced Traffic Detection". In: *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*. IEEE. 2018, pp. 208–215 *Cited on page 62.*

[75] Megan Thomas and Ellen W Zegura. *Generation and analysis of random graphs to model internetworks*. Tech. rep. Georgia Institute of Technology, 1994 *Cited on page 65.*

[76] OSBoxes. *Android x86*. URL: https://www.osboxes.org/android-x86/ *Cited on page 72.*

[77] Lara López et al. "Heterogeneous secure multi-level remote acceleration service for low-power integrated systems and devices". In: *Procedia Computer Science* 97 (2016), pp. 118–121 *Cited on page 73.*

[78] Fotis Foukalas et al. "Protocol reconfiguration using component-based design". In: *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2005, pp. 148–156 *Cited on page 73.*

[79] Jae Woo Lee et al. "z2z: Discovering zeroconf services beyond local link". In: *2007 IEEE Globecom Workshops*. IEEE. 2007, pp. 1–7 *Cited on page 74.*

[80] Houssemeddine Mazouzi, Nadjib Achir, and Khaled Boussetta. "Elastic Offloading of Multitasking Applications to Mobile Edge Computing". In: *Proceedings of the 22st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2019, Miami Beach, FL, USA, November*

*25 - November 29, 2019*. 2019, pp. 1–8. DOI: 10.1145/3345768.3355926 *Cited on pages 97, 127.*

[81] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455 *Cited on page 108.*

[82] H. Wang and O. Sinnen. "List-Scheduling versus Cluster-Scheduling". In: *IEEE Transactions on Parallel and Distributed Systems* 29.8 (Aug. 2018), pp. 1736–1749 *Cited on page 109.*

[83] Nancy M Amato and Ping An. "Task scheduling and parallel mesh-sweeps in transport computations". In: *TR00-009, Department of Computer Science, Texas A&M University* (2000) *Cited on page 112.*

[84] Huaming Wu et al. "A Novel Offloading Partitioning Algorithm in Mobile Cloud Computing". In: *Unpublished* (2015) *Cited on page 113.*

[85] P ERDdS and A R&wi. "On random graphs I". In: *Publ. Math. Debrecen* 6 (1959), pp. 290–297 *Cited on page 116.*

**Titre :** **Algorithmes pour le déchargement de tâches sur serveurs de périphérie**

**Mots clefs :** déchargement de calculs, cloud, mobile edge computing, terminal mobile, partitionnement d'application, décomposition de Lagrange, graphe de tâches, DAG.

**Résumé :** Le déchargement de calculs est l'une des solutions les plus prometteuses pour surmonter le manque de ressources au niveau des terminaux mobiles. Elle permet l'exécution d'une partie ou de la totalité d'une application mobile dans le cloud. L'objectif est d'améliorer les temps d'exécution et de réduire la consommation énergétique. Malheureusement, le cloud est généralement éloignés des équipements terminaux. Ce qui rend cette approche souffrir de délais importants et fluctuants. Cela est particulièrement problématique pour certaines applications pour lesquelles un temps de réponse réduit est nécessaire. Pour réduire ce délai d'accès, l'une des approches émergentes est de pousser le Cloud vers la bordure du réseau. Cette proximité permet aux applications mobiles de décharger leurs tâches et données vers un Cloud "**local**" ou "Edge Cloud".
Dans cette thèse, nous nous concentrons sur le déchargement de calculs dans une architecture de type mobiles (**Mobile Edge Computing** – MEC), composée de plusieurs serveurs de périphérie. Notre objectif est d'explorer de nouvelles stratégies de déchargement efficaces afin d'améliorer les performances des applications tant du point de vue délais de calcul que consommation énergétique, tout en garantissant les contraintes de temps d'exécution des applications. Notre première contribution est une nouvelle stratégie de déchargement sur plusieurs serveurs de périphérie. Nous proposons par la suite une extension de cette stratégie en incluant également le Cloud. Nous évaluons ces stratégies tant du point de vue théorique que pratique avec l'implémentation d'un middleware de déchargement. Finalement, nous proposons une nouvelle approche élastique dans le cas d'applications multitâches caractérisées par un graphe de dépendances entre les tâches.

**Title :** **Algorithms for Tasks Offloading on Multiple Mobile Edge Servers**

**Keywords :** computation offloading, cloud, mobile edge computing, mobile terminal, application partitioning, Lagrangian decomposition,task graph, DAG.

**Abstract :** Computation offloading is one of the most promising new paradigm to overcome the lack of computational resources in mobile devices. Basically, it allows the execution of part or all of a mobile application in the cloud. The main objective is to reduce both execution time and energy consumption for the mobile terminals. Unfortunately, even if clouds have rich computing and storage resources, they are usually geographically far from mobile applications and may suffer from large delays, which is particularly problematic for mobile applications with small response time requirements. To reduce this long delay, one of the emerging approach is to push the cloud to the network edge. This proximity gives the opportunity to mobile users to offload their resources consuming tasks to "**local**" cloud for processing. An **Edge Cloud** can be seen as small data center acting as a shadow image of larger data centers. This geographical proximity between mobile applications and edge cloud means that the access delay can be greatly reduced, but also higher throughput, a greater responsiveness and better scalability.
In this thesis, we focus on computation offloading in mobile environment (**Mobile Edge Computing** - MEC), composed of several edge servers. Our goal is to explore new and effective offloading strategies to improve application performance in both execution time and energy consumption, while ensuring application requirements. Our first contribution is a new offloading strategy in the case of multiple edge servers. Then we extend this strategy to include the Cloud. Both strategies have been evaluated theoretically and experimentally by the implementation of an offloading middleware. Finally, we propose a new elastic approach in the case of multitasking applications characterized by a graph of dependencies between the tasks.

***École doctorale Galilée, Université Paris 13,***
***Laboratoire de Traitement et Transport de l'Information (L2TI).***
*Villetaneusse, France*