User equipment based-computation offloading for real-time applications in the context of Cloud and edge networks

Farouk Messaoudi

ANNÉE 2018

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MathStic

Farouk MESSAOUDI

Préparée à l'unité de recherche (IRISA – UMR 6074)
Institut de Recherche en Informatique et Système Aléatoires
Université de Rennes 1

User equipment based computation Offloading for real-time applications in the context of Cloud and edge networks

Thèse à soutenir à Rennes

le 16 avril 2018

devant le jury composé de :

Houda LABIOD
Professeur, Telecom ParisTech / *examinateur*
Riccardo SISTO
Professeur, École polytechnique de Turin / r*apporteur*
Toufik AHMED
Professeur, Bordeaux INP / *rapporteur*
César VIHO
Professeur, Université de Rennes1 / *examinateur*
Erwan LEMERRER
Docteur, Technicolor / *examinateur*
Philippe BERTIN
Manager de projets, Orange Labs / *co-directeur de thèse*

ABSTRACT

Computation offloading is a technique that allows resource-constrained mobile devices to fully or partially offload a computation-intensive application to a resourceful Cloud environment. Computation offloading is performed mostly to save energy, improve performance, or due to the inability of mobile devices to process a computation-heavy task. There have been a numerous approaches and systems on offloading tasks in the classical Mobile Cloud Computing (MCC) environments such as, CloneCloud, MAUI, and Cyber Foraging. Most of these systems are offering a complete solution that deal with different objectives. Although these systems present in general good performance, one common issue between them is that they are not adapted to real-time applications such as mobile gaming, augmented reality, and virtual reality, which need a particular treatment.

Computation offloading is widely promoted especially with the advent of Mobile Edge Computing (MEC) and its evolution toward Multi-access Edge Computing which broaden its applicability to heterogeneous networks including WiFi and fixed access technologies. Combined with 5G mobile access, a plethora of novel mobile services will appear that include Ultra-Reliable Low-latency Communications (URLLC) and enhanced Vehicle-to-everything (eV2X). Such type of services requires low latency to access data and high resource capabilities to compute their behaviour.

To better find its position inside a 5G architecture and between the offered 5G services, computation offloading needs to overcome several challenges; the high network latency, resources heterogeneity, applications interoperability and portability, offloading frameworks overhead, power consumption, security, and mobility, to name a few.

In this thesis, we study the computation offloading paradigm for real-time applications including mobile gaming and image processing. The focus will be on the network latency, resource consumption, and accomplished performance.

The contributions of the thesis are organized on the following axes:

- Study game engines behaviour on different platforms regarding resource consumption (CPU/GPU) per frame and per game module.
- Study the possibility to offload game engine modules based on resource consumption, network latency, and code dependency.
- Propose a deployment strategy for Cloud gaming providers to better exploit their resources based on the variability of the resource demand of game engines and the QoE.
- Propose a static computation offloading-based solution for game engines by splitting 3D world scene into different game objects. Some of these objects are offloaded based on resource consumption, network latency, and code dependency.
- Propose a dynamic offloading solution for game engines based on an heuristic that compute for each game object, the offloading gain. Based on that gain, an object may be offloaded or not.
- Propose a novel approach to offload computation to MEC by deploying a mobile edge application that is responsible for driving the UE decision for offloading, as well as propose two algorithms to make best decision regarding offloading tasks on UE to a server hosted on the MEC.

RÉSUMÉ

Le délestage de calcul ou de code est une technique qui permet à un appareil mobile avec une contrainte de ressources d'exécuter à distance, entièrement ou partiellement, une application intensive en calcul dans un environnement Cloud avec des ressources suffisantes. Le délestage de code est effectué principalement pour économiser de l'énergie, améliorer les performances, ou en raison de l'incapacité des appareils mobiles à traiter des calculs intensifs. Plusieurs approches et systèmes ont été proposés pour délester du code dans le Cloud tels que CloneCloud, MAUI et Cyber Foraging. La plupart de ces systèmes offrent une solution complète qui traite différents objectifs. Bien que ces systémes présentent en général de bonnes performances, un problème commun entre eux est qu'ils ne sont pas adaptés aux applications temps réel telles que les jeux vidéo, la réalité augmentée et la réalité virtuelle, qui nécessitent un traitement particulier.

Le délestage de code a connu un récent engouement avec l'avènement du MEC et son évolution vers le edge à multiple accès qui élargit son applicabilité à des réseaux hétérogènes comprenant le WiFi et les technologies d'accès fixe. Combiné avec l'accès mobile 5G, une pléthore de nouveaux services mobiles apparaîtront, notamment des service type URLLC et eV2X. De tels types de services nécessitent une faible latence pour accéder aux données et des capacités de ressources suffisantes pour les exécuter.

Pour mieux trouver sa position dans une architecture 5G et entre les services 5G proposés, le délestage de code doit surmonter plusieurs défis; la latence réseau élevée, hétérogénéité des ressources, interopérabilité des applications et leur portabilité, la consommation d'énergie, la sécurité, et la mobilité, pour citer quelques uns.

Dans cette thèse, nous étudions le paradigme du délestage de code pour des applications a temps rèel, par exemple; les jeux vidéo sur équipements mobiles et le traitement d'images. L'accent sera mis sur la latence réseau, la consommation de ressources, et les performances accomplies.

Les contributions de la thse sont organisées sous les axes suivants:

- Étudier le comportement des moteurs de jeu sur différentes plateformes en termes de consommation de ressources (CPU / GPU) par image et par module de jeu.
- Étudier la possibilité de distribuer les modules du moteur de jeu en fonction de la consommation de ressources, de la latence réseau, et de la dépendance du code.
- Proposer une stratégie de déploiement pour les fournisseurs de jeux dans le Cloud, afin de mieux exploiter les ressources, en fonction de la demande variable en ressource par des moteurs de jeu et de la QoE du joueur.
- Proposer une solution de délestage statique de code pour les moteurs de jeu en divisant la scène 3D en différents objets du jeu. Certains de ces objets sont distribués en fonction de la consommation de ressources, de la latence réseau et de la dépendance du code.
- Proposer une solution de délestage dynamique de code pour les moteurs de jeu basée sur une heuristique qui calcule pour chaque objet du jeu, le gain du délestage. En fonction de ce gain, un objet peut être distribué ou non.
- Proposer une nouvelle approche pour le délestage de code vers le MEC en déployant une application sur la bordure du réseau (edge) responsable de la décision de délestage au niveau du terminal et proposer deux algorithmes pour prendre la meilleure décision concernant les tâches à distribuer entre le terminal et le serveur hébergé dans le MEC.

ACKNOWLEDGEMENTS

Dans cette thèse, je présente le travail que j'ai effectué sur une période de plus de trois ans. Pendant longtemps, de nombreuses personnes m'ont aidé, m'ont inspiré, ou autrement m'ont soutenu pour arriver là où je suis aujourd'hui. Ces gens que je voudrais remercier ici.

Tout d'abord, je remercie tout particulièrement mon directeur de thèse Adlen Ksentini et co-directeur Philippe Bertin, pour leurs conseils, leur disponibilité et dévouement, leur gentillesse, leur sympathie et surtout de m'avoir accordé leur confiance pour mener à bien ce travail; qu'ils puissent trouver ici l'expression de ma reconnaissance, de mon profond respect et de mes plus vifs remerciements.

Au terme de ce travail, il m'est agréable de remercier toutes les personnes qui, chacune à sa manière, m'ont permises de le mener à bien.

Tout d'abord, mon chef de laboratoire "Network Architecture" Mr Olivier Choisy.

Aussi, je remercie infiniment mon directeur du lab Mr Michel Corriou.

Un grand merci aux employés de B-COM pour leur disponibilité.

Je remercie les membres du jury d'avoir accepté de juger ce travail.

Enfin, que tous ceux qui ont participé de près ou de loin à la bonne marche de ce travail et la réalisation de ce mémoire trouvent ici l'expression de ma reconnaissance et de mes remerciements les plus profonds.

À toutes et à tous Merci.
Farouk

TABLE OF CONTENTS

LIST OF TABLES

TABLE **Page**

LIST OF FIGURES

FIGURE **Page**

GLOSSARY

AI Artificial Intelligence

AIFF Audio Interchange File Format

AMBR Aggregate Maximum Bit Rate

AP Access Point

APN Access Point Name

API Application Programming Interface

AR Augmented Reality

ARP Allocation and Retention Priority

ASCII American Standard Code for Information Interchange

BSR Buffer Status Report

CAARS Context Aware-Augmented Reality System

CC Cloud Computing

CDF Cumulative Distribution Function

CDN Content Delivery Network

CGP Cloud Gaming Provider

CISC Complex Instruction Set Computer

CPU Central Processing Unit

CQI Channel Quality Indicator

CSI Channel State Information

C-RNTI Cell-Radio Network Temporary Identifier

DAC Dynamic Audio-Visual Communication

DSCP DiffServ Code Point

DNS Domain Name System

DVS Dynamic Voltage Scaling

eNB eNodeB

EPC evolved Packet Core

EPS evolved Packet System

ETSI European Telecommunications Standards Institute

EWMA Exponential Weighting Moving Average

fps Frames Per Second

FMC Follow me Cloud

FPS First Person Shooter

GBR Guaranteed Bit Rate

GNSS Global Navigation Satellite System

GO Game Object

GOAP Goal Oriented Action Planning for a smarter AI

GPRS General Packet Radio Service
GPS Global Positioning System
GPU Graphic Processing Unit
GUI Graphical User Interface
GVSP GigE Vision Stream Protocol
HCI Human Computer Interface
HD High-Definition
HDR High Dynamic Range
HFR High Frame Rate
HID Human Interface Device
HTTP Hypertext Transfer Protocol
IaaS Infrastructure as a Service
iSCSI internet Small Computer System Interface
InPs Infrastructure Providers
ID Identifier
ILP Integer Linear Programming
IP Internet Protocol
IQR Inter-Quartile Range
IVE Immersive Virtual Environments
I/O Inputs/Outputs
IoT Internet of Things
jSLP java Service Location Protocol
JDK Java Development Kit
JVM Java Virtual Machine
KPI Key Performance Indicator
LAN Local Area Network
LISP Locator Identifier Separation Protocol
LOD Level of Details
LTE Long Term Evolution
MA Mobile Agents
MAC Medium Access Control
MAN Metropolitan Area Network
MBR Maximum Bit Rate
MCC Mobile Cloud Computing
MDP Markovian Decision Process
ME Mobile Edge

MEC Mobile Edge Computing

MIMO Multiple Inputs Multiple Outputs

MIMD Multiple Instructions Multiple Data

MME Mobility Management Entity

MMOG Massively Multi-player Online Games

MMORPG Massively Multi-player Online Role-Playing Games

MMU Memory Management Unit

MNO Mobile Network Operator

MPTCP Multipath TCP

NDK Native Development Kit

NFV Network Function Virtualization

NIST National Institute of Standards and Technology

NPC Non-Player Character

ODE Open Dynamics Engine

OFDMA Orthogonal Frequency Division Multiplexing

OOP Oriented-Object Programming

OS Operating System

OSS Operating Support System

PaaS Platform as a Service

PC Player Character

PCRF Policy and Charging Resource Function

PDA Personal Digital Assistant

PDN Packet Data Network

PLMN ID Public Land Mobile Network Identifier

PRB Physical Resource Block

QAM Quadrature Amplitude Modulation

QCI QoS Class Identifier

QoE Quality of Experience

QoS Quality of Service

QPSK Quadrature Phase-Shift Keying

RAN Radio Access Network

RAM Random Access Memory

RB Resource Block

RBS Remaining Block Signal

RIM Research in Motion

RISC Reduced Instruction Set Computer

RMI Remote Methods Invocation

RNIS Radio Network Information Service

RPC Remote Procedure Call

RPG Role-Playing Game

RTP Real-Time Transport Protocol

RTS Real-Time Strategy

RTT Round Trip Time

RVLG Reduced Valued Locality-Labeled Graph

SaaS Software as a Service

SCC Strongly Connected Components

SDK Software Development Kit

SDN Software-Defined Networking

SIMD Single Instruction Multiple Data

SOA Service Oriented Architecture

SLA Service-Level Agreement

SLP Service Location Protocol

SNR Signal to Noise Ratio

SOSGI Single Online Shared Game Instance

TCP Transmission Control Protocol

TPS Third Person Shooter

TSP Telecommunication Service Provider

TTI Transmission Time Interval

UDDI Universal Description Discovery and Integration

UDP User Datagram Protocol

UE User Equipment

UI User-Interface

UMSC Universal Mobile Service Cell

UPnP Universal Plug and Play

URL Uniform Resource Locator

VIM Virtualization Infrastructure Manager

VLG Valued Locality-Labeled Graph

VM Virtual Machine

VMM Virtual Machine Manager

VOIP Voice over IP

VPN Virtual Private Network

VR Virtual Reality

VSM Virtual Server Manager

WAN Wide Area Network

Wi-Fi Wireless-Fidelity

XML Extensible Markup Language

3G 3^{rd} Generation

5G 5^{th} Generation

3GPP 3^{rd} Generation Partnership Project

CHAPTER 1

INTRODUCTION

Nowadays, mobile devices such as smartphones and tablets, are gaining enormous popularity. Smartphones are ubiquitous devices, offering "information at your fingertips" ranging from communications to computing services. These pervasive devices have created an acute dependency among mobile users, many of whom cannot leave home without taking their device. Mobile users expect to be able to run computationally intense, high quality applications on these devices, which include mobile-gaming (m-gaming), mobile-health (m-health), and mobile-learning (m-learning) applications. However, due to the lightness, handiness, and compactness of these devices, and due to users safety and mobility issues, mobile devices have some intrinsic limitations to their capabilities, which include low processing ability, short battery autonomy, small storage capacity, and limited screen size. These limitations impede the execution of resource-intensive applications. To make this feasible, industry and academia have two options;

- ***Hardware augmentation***. This consists of enhancing the hardware capabilities of mobile devices, which include the CPU, GPU, battery, and storage.

- ***Software augmentation***. This leverages remote infrastructure resources to offload the intensive computation in order to conserve local resources.

Several investigations have been carried out to increase the hardware capabilities of mobile devices. Manufacturers are proposing multi-core processors with high clock speed, and the ARM industry offers various microprocessor cores that address performance, power and cost requirements, such as the *Cortex-A* processor, which integrates a Memory Management Unit (MMU), designed to execute complex

operating systems, including Linux, Android, and Microsoft Windows[1]. Samsung[2] and Qualcomm[3] corporations have designed new generations of smartphones, which include GPU's, such as *Mali* and *Adreno*. In regards to battery autonomy, many efforts are being made to harvest energy from renewable resources, including human movement [211], solar energy [23, 172], and wireless radiation [113]. These solutions are still under investigations and currently cannot noticeably enhance the energy source, as these resources are intermittent and not available on-demand. In parallel, researchers are attempting to reduce the energy overhead in different aspects of computing, including hardware, Operating Systems (OSs), applications, and networking interfaces. Dynamic Voltage Scaling (DVS) technology is a power management technique used to conserve energy, particularly in wearable devices, by increasing and decreasing the voltage as needed. The automatic screen shutdown is another attempt to save energy on mobile devices. For some companies, like Samsung, the removable battery is a good way to keep the smartphone working as long as possible by changing the spent battery out for another one.

Software augmentation has also attracted industrial and academic solutions, which conserve the local resources of mobile devices. Mobile Cloud Computing is envisioned as a promising solution for addressing the aforementioned challenges. By offloading computations to resourceful servers located in the Cloud, Mobile Cloud Computing can augment the capabilities of mobile devices for computationally-hungry applications. Several approaches have been proposed in the literature based on the Mobile Cloud Computing; we cite load sharing, remote execution, and computation offloading. Load sharing and remote execution have evolved to the concept of computation offloading, which is more general and mature. Computation offloading has become an attractive way to reduce the execution time required by mobile users. It consists of moving a portion of an application to the Cloud. Upon receiving the migration request, the server creates a dedicated virtual machine for the mobile device, loads the application, and starts its execution. In the mean time, the mobile device continues to run other tasks or waits for the results of the execution on the server. At the end, the migrated portion returns to the mobile device, and merges back into the original process. Computation offloading involves various stakeholders: mobile devices, network environment, mobile applications, remote servers, mobile users, and the framework performing the migration.

Accordingly, several challenges and issues have arisen from different stakeholder perspectives that need to be addressed, for example, response time, energy consumption, security, network communications, and network latency. Research has been done to address the computation offloading from these points of view. Some of these challenges have been solved, such as the network latency, wherein Mobile Edge Computing and Fog Computing have been widely used in computation offloading to address this issue. In terms of network communications, multi-Radio Access Technology (multi-RAT) and on-demand bandwidth have been used to improve the throughput and, therefore, improve the offloading performance. Some other challenges are still open issues, such as the framework overhead, and decision-making

[1] `https://www.arm.com/products/processors`
[2] `http://www.samsung.com/semiconductor/minisite/exynos/`
[3] `https://www.qualcomm.com/products/mobile-processors`

regarding when and what to offload. This thesis will address some of these issues.

1.1 Thesis Contributions

The general thematic of the thesis concerns the computation offloading of real-time, resource-intensive, and complex mobile applications, including 3D game engines and facial recognition. The research topics are based on feasibility, performance, and network latency.

The contributions of the thesis are three-fold:

1. Game engines: Under this axis, there are two contributions [189, 191]. This axis can split into three parts:

   a) Deployment strategy: Game engines are studied in the Cloud gaming context in terms of resource variability. Different types of games have been tested, and the obtained performance are considered for server consolidation. Cloud gaming providers should find a trade-off between the user's Quality of Experience (QoE) and the number of hosted game engines per server. Currently, Cloud servers are being split among several game engines using virtualization technology. The higher the resource demand of a game engine, the more resourceful the virtual machine hosting this game engine will be, resulting in low server consolidation. If the resource demands of a game engine do not vary by much, a Cloud gaming provider can easily predict this demand, and, therefore, consolidate the server efficiently. However, if the game engine varies widely in its demand for resources, then, in such a case, a Cloud provider should accommodate the game engine with peaks. Therefore, a Cloud gaming provider should find a trade-off between high consolidation and high QoE, using the load variability study in [189].

   b) Game engine dissection: Game engines, to our knowledge, have not being studied from the state-of-the-art internal architecture point of view. Game engines are considered as proprietary black boxes. Considering the recent trends, game players have become more mobile and are using their smartphones and tablets to play games. Compared to dedicated boxes, such as the XBox One, mobile devices do not perform well when running best-selling 3D games. Solution leveraging remote resourceful infrastructure, such as Cloud gaming and computation offloading, represent the next step toward improving the gaming experience. Consequently, dissecting, testing, and analysing game engine behaviour is a step toward better understanding how to distribute game engines over a network. Therefore, this contribution aims to analyse the behaviour of different "genres" of games, test their performance, identify resource bottlenecks, and, most importantly, draw the internal call flows for inter- and intra-modules that compose a game.

   c) Offloading feasibility: For each module of the game engine, we represent the internal call flow in order to study the offloading case. Modules related to rendering, which represent

the majority of the CPU consumption and most of the GPU consumption are all mixed by series of calls. This observation is a challenge for the implementation of code offloading since it reduces the gains in terms of performance, and it imposes to generate calls between distant machines. The physics family can be spread into three sub-families that communicate through one class. It would thus be relatively simple to distribute the computation of these three sub-families over separate computers. The scripting family is the only module that interacts directly with the physics engine. It would thus make sense to run this family of modules on the same computer as physics modules. The audio family has no interaction with the other main modules and only deals with the game thread. This module can thus be offloaded as an Application Programming Interface (API) to a remote machine. The communication between the client and the remote machine can be done either by Remote Procedure Call (RPC) or by streaming the audio data. The Artificial Intelligence (AI) family can be computationally intensive. It is difficult or even impossible to simulate the game on constrained-resource devices without reducing the complexity of the AI. Since this family is also mostly independent, the module family as a whole can be offloaded without introducing much complexity.

2. Game engine offloading: Under this axis, there are two contributions [187, 189]. In [189], a static offloading of game engine modules based on game engine objects has been proposed. We decomposed a game world into several game objects; these include the player character, non-player character, and the environment. We set several criteria regarding these objects. The criteria are based on resource consumption, code dependency constraints, network latency, and bandwidth consumption. According to these criteria, each game object is placed on a mobile device or a remote server. In [187] a dynamic offloading of the game engine modules has been proposed. This contribution is an enhancement of our work done in [189], wherein we proposed an heuristic to schedule module placement using the same criteria.

3. MEC oriented offloading contribution: We have a contribution under this axis [188]. One of the big concerns in computation offloading is the network latency between the mobile device and server, which limits computation offloading applicability to delay-tolerant applications. For delay-sensitive applications such as gaming, high latency is intolerable and degrades the quality as perceived by users. Hopefully, with Mobile Edge Computing (MEC) and the transition toward 5G, network latency will be drastically reduced. Our contribution was to leverage MEC architecture to drive computation offloading. We want to use MEC as an enabler for low-latency, computation offloading-based applications, such as facial recognition. To this aim, we designed a framework to orchestrate the offloading process. The framework runs on three different entities: (i) *mobile edge application* hosted on the mobile edge, which is able to access mobile user-related radio information, (ii) *mobile user*, and (iii) a *server* hosted in the mobile edge. The mobile edge application is responsible for driving a decision to accept or reject offloading requests coming

from mobile devices and predicting, using low-level API, the latency value. The mobile device makes the decision to offload application modules or not according to the estimated latency value obtained from the mobile edge application. Finally, the server, hosted on the mobile edge side, computes the offloaded code and sends back the results to the mobile device. As a proof of concept, We have run a facial recognition application on a mobile device which offloads computation to the server hosted in the mobile edge.

1.2 Manuscript Organization

Chapter 2 introduces the reader to computation offloading. We provide a definition of computation offloading in the frame of past studies. We identify the main steps for performing a computation offloading request based on past studies. Then we match these steps in a use case based on Google glasses. Next, we classify the state-of-the-art computation offloading into three approaches based on timescale decision making. We also provide an evolutionary history of computation offloading according to the literature. Further, we introduce the metrics that impact computation offloading performance, which will become the research topic in chapter 3. We finish this chapter by describing the advantages and disadvantages of computation offloading.

Chapter 3 presents, comprehensively and qualitatively, an exhaustive state-of-the-art computation offloading from six points of view. From the mobile user perspective, we focus on the objective function behind computation offloading desired by the mobile user. Mobile users are particularly interested in the performance and energy consumption of their mobile applications. We classified several frameworks according to the objective function into mono-objective, bi-objective, and multi-objective optimization-based frameworks. Then we discuss user mobility from a computation offloading context, and we finish with some user concerns regarding computation offloading. From the mobile device perspective, we present the various platforms used in the literature for computation offloading and the mobile device's hardware evolution. Next, from the server side, we describe each of the server types, platform types, improvements, and issues regarding computation offloading. All of these axes are based on past studies. A section of this chapter is also dedicated to describe computation offloading from an application point of view. We describe the taxonomy regarding the "genre" of applications, the programming languages used to develop these applications, the requirements, and the dependency of the applications code on the underlying platform and hardware. Computation offloading relies on network communication, therefore, it seems adequate to describe computation offloading from the network perspective, especially the metrics, communications support, and network models. We then highlight the steps involved in offloading a computation, data availability, framework location, framework scale, application annotations, offloading types, and the models and algorithms devised to steer computation offloading decision making.

Chapter 4 focuses on game engines from a computation offloading perspective. In this chapter, a background of game engines that describes the architecture with the associated modules, and the process of generating frames done under the graphic pipeline is presented. The background also introduces three

game engine solutions that eliminate the powerless capabilities of mobile gamers. The solutions are Cloud gaming, computation offloading, and client-server architecture, each of which presents some challenges regarding the "genre" of games, such as First Person Shooter (FPS), Real-Time Strategy (RTS), or Massively Multi-player Online Games (MMOG). Once the game engine's landscape is drawn, we present our methodology for classifying game engines under the three solutions cited above. The classification is based on the variability and playability of game engines, for which we define predicates. The chosen criteria are central to the Cloud gaming provider being able to manage the Quality of Service/Quality of Experience (QoS/QoE) and consolidation, but also to mobile user immersion and interactivity. Next, we dissect the internal architecture and call flow in the different modules. The aim is to study the possibility of offloading software components of these modules. We finish the chapter by demonstrating our classification via use cases of running games under the three solutions.

Chapter 5 describes a novel solution for mobile gaming based computation offloading. We begin this chapter by reviewing past studies of game engines-oriented computation offloading. Almost all of this research presents the same challenges; the real-time constraints and high-resource consumption of 3D best-selling FPS games. Our solution resolves these problems. We described how a 3D scene is represented in a game engine and what the patterns and objects are in a game world. Then, we present a vision and methodology to design the 3D-based Unity Computation Offloading Framework. We propose an heuristic based on network communications to select the objects with the associated modules that should be offloaded. We close the chapter by testing our framework from the resource consumption, network communications, responsiveness, and frame rate perspectives.

Chapter 6 mainly focuses on the MEC. We saw in previous chapters that latency is central in computation offloading, particularly for real-time applications. For this aim, we bring computation resources closer to mobile users. In particular, we leverage MEC to perform computation offloading for facial recognition using a case study. This chapter begins by presenting the MEC, its architecture, and describing the technical terms used throughout the chapter, particularly those related to the MEC and LTE communications. Next, we review state-of-the-art, MEC-oriented computation offloading. After that, we propose our framework, which relies on MEC, its services, and APIs to derive the user's equipment offloading decision. The framework is composed of three entities: middleware on both user equipment and the MEC server, and a mobile edge application that is responsible for the decision making in terms of computation offloading. The framework is then tested in a facial recognition use case.

Finally, concluding remarks are made in Chapter 7, and future directions and perspectives are discussed.

CHAPTER 2

COMPUTATION OFFLOADING: CONTEXT

2.1 Introduction

Nowadays, mobile devices, such as smartphones and tablets, have become ubiquitous. Due to the incorporation of various captors and hardware functionality in one small box, they have become a constant fixture in many lives. This has increased the demand on the part of mobile users for computationally intensive applications, such as m-gaming, m-health, and m-learning. However, as far as we know, the capabilities of these mobile devices are very much constrained, creating a gap between performance and application requirements. A promising solution with which to extend the capabilities of these mobile devices is to leverage Cloud infrastructure or any resourceful platform to perform the computation, fully or partially, for the said applications. This solution is known as *computation offloading*.

Computation offloading not only saves energy and improves performance, it also makes execution of heavy-complex applications on powerless devices possible. A great deal of research has been done on computation offloading, such that of Mobile Assistance Using Infrastructure (MAUI in short) [59], Scavenger [147], and Spectra [81]. These examples, and others that we will review in this manuscript, seem to diverge on some criteria and converge on others. To better understand these works and the computation offloading concept, in general, we propose to review state-of-the-art computation offloading to understand the concept. In this chapter, we define the concept of computation offloading in the frame of the related work (e.g., [59]), and extract some keystone similarities between most offloading frameworks, which are organized into several steps; profiling, modelling, partitioning, and communications. The aim is to propose a template to follow in order to design applications with certain elasticity and to improve the offloading performance via enhancement of these keystones. Then, we classify the computation offloading approaches into three types depending on when the aforementioned keystones are performed. We also provide the historic evolution of the computation offloading concept. Afterwards, we focus on

the relevance of several metrics in terms of computation offloading feasibility, performance, and gain.

Above all, we present the main advantages and disadvantages of computation offloading, and finally, we summarize, and introduce the next chapter.

2.2 Principals

The concept of computation offloading has been introduced in many forms over the years. Accelerated by the emergence of Cloud computing, virtualization technology advances, and wireless network technology improvements (including, mobile broadband), computation offloading attracted a tremendous amount of research to make it a reality. Besides reducing energy consumption [166, 168], computation offloading allows powerless devices (e.g., smartphones) to run CPU-intensive applications, such as 3D gaming [192], facial recognition [249], and 3D rendering [200]. To clearly cover the topic, we present hereafter the decomposition of the concept into different steps followed by the offloading types according to timescale, and then, the history of the computation offloading concept.

2.2.1 Definition

Usually, computation offloading refers to the transfer of certain [59, 91] or all [114, 249] computing tasks of an application to an external platform, which includes clusters [21], grids [96] or the Cloud [59], in order for these to be executed there. The purpose behind this transfer is to extend the capabilities of today's powerless mobile devices and increase their batteries' autonomy. Upon successful execution, the results are sent back to the mobile device to be reintegrated with the application. To ensure a better harnessing of the computation offloading concept, it is necessary to design algorithms that decide which tasks of an application should be offloaded and when. Overall, most computation offloading frameworks rely on four steps, which we provide in detail below:

1. Application Profiling. This step consists of dissecting the application structure to estimate resource consumption investigating such things as processor time, energy consumption, and memory utilization. A well-known technique for recording program behaviour and measuring its performance is to insert code into the program [19, 80, 224]. The granularity level of this estimation (e.g., class- or function-level) is determined by taking into account performance overheads and privacy concerns. Several systems and frameworks have been devised for computation offloading with dedicated profilers. For instance, *MAUI* [59] evaluates the energy consumption of each method (i.e., function, procedure), and *ThinkAir* [139] estimates the CPU cost, memory utilization, latency, and bandwidth utilization. For the same purpose, *Scavenger* [147] uses a dual, adaptive, history-based profiling approach to evaluate the execution time of an application, and *Spectra* [81] profiles the CPU time, network, power consumption, cache utilization, remote CPU consumption, and remote cache state for each task.

2. Application Modelling. Usually, in this step, we define a model to describe the problem based on the granularity defined in the precedent step. Most existing works are related to the methodologies used to

identify the parts of the application that can be offloaded. Some works represent the application with a valued graph [100], wherein the vertices correspond to the application components, valued with the resource consumption metrics of these components, while the edges represent the interactions between the components, valued with the frequency of calls and the amount of the exchanged data between the calls. The optimal solution is obtained with Integer Linear Programming (ILP) models [143]. Other studies rely on Markov theory [41] to design the mobile device platform with queues and Markovian Decision Processs (MDPs). Recent studies follow a game-theoretic approach to represent the application as a game between selfish players, in which equilibrium should be derived to satisfy each player [34].

3. Application Partitioning. In this step, a decision is made for each component as to whether it should be offloaded or not. The decision is based on an objective function, such as improving performance or saving energy, that was assigned to the model in the previous step. The objective function may be resolved at run-time, which known as *dynamic offloading* [59], or prior to the execution of the application, which is known as *static offloading* [32]. Many algorithms have been proposed to solve the objective function. For instance, graph partitioning is done by applying graph partitioning algorithms to the resulting graph. Often, this is accomplished using maximum flow-minimum cut algorithms, such as the pre-flow-push [100] or the Stoer-Wagner [209]. The aim of such algorithms is to partition the graph representing the application into a minimum cut, which includes the local components in one partition, and the components, being considered for remote execution in the second partition. Since graph partitioning is NP-hard, heuristics can be used to efficiently approximate solutions when dealing with large graphs using branch and bound [166], greedy [231], or genetic algorithms [63].

4. Communication. The last step consists in establishing a communications channel between the client and server to migrate the code and required data from the mobile device to the server and back again with the results of the computation. Various mechanisms have been used for communications in a client-server model. Some frameworks, such as Chroma [18] and Spectra [82], use Remote Procedure Call (RPC), wherein the tasks are pre-installed on the server and ready for service. Thus, the desired functionality is always available on the server, and there is no need to install it on -demand. Despite their good performance, RPC solutions cannot be applied to a mobility environment which does not support RPC. Similar to RPC, Remote Methods Invocation (RMI) can be used for remote calls [84] between Java Virtual Machines (JVMs). The Java objects are automatically serialized and sent to the remote JVM. In Uniform Resource Locator (URL)-based download approaches [93], instead of migrating the code, an URL is sent to the server, indicating the location of the code to download.

To illustrate the aforementioned steps, we present the case described by Satyanarayanan et al. [249] in 2009.

> *Ron has recently been diagnosed with Alzheimer's disease. Due to the sharp decline in his mental acuity, he is often unable to remember the names of friends and relatives; he also frequently forgets to do simple daily tasks. He faces an uncertain future that is clouded by a lack of close family nearby and limited financial resources for professional caregivers.*

> *Even modest improvements in his cognitive ability would greatly improve his quality of life, while also reducing the attention demanded from caregivers. This would allow him to live independently in dignity and comfort for many more years, before he has to move to a nursing home. Fortunately, a new experimental technology might provide Ron with cognitive assistance. At the heart of this technology is a lightweight wearable computer with a head-up display in the form of eyeglasses. Built into the eyeglass frame are a camera for scene capture and earphones for audio feedback. These hardware components offer the essentials of an augmented-reality system to aid cognition when combined with software for scene interpretation, facial recognition, context awareness, and voice synthesis. When Ron looks at a person for a few seconds, that person's name is whispered in his ear along with additional cues to guide Ron's greeting and interactions; when he looks at his thirsty houseplant, "water me" is whispered; when he looks at his long-suffering dog, "take me out" is whispered.*

The eyeglasses architecture is composed of different applications, including Augmented Reality (AR), computer vision, and audio processing to serve different needs for users. Due to the resource-intensive nature of these applications and the intrinsic limitations of eyeglasses, offloading computationally-intensive functionality is needed to meet applications requirements, particularly for the delay-sensitive applications, such as AR [280].

Figure 2.1 portrays the aforementioned offloading steps applied to a simplified version of Google' glasses [66]. The system software architecture is made up of several services and subsystems: The visual subsystem is in charge of processing image recognition and facilitating real-world recognition. The Human Computer Interface (HCI) subsystem is responsible for speech recognition and speech synthesis to allow natural interaction with the different learning interfaces. The Training subsystem is the logical entity which can host new functionality for future usage. The glasses have two main limitations: low computing capabilities and battery drains due to the high utilization of hardware resources, including the miniature camera, heads-up display, and mini-speakers. A solution is to offload the CPU-intensive functions to close-by, powerful, plugged machines (e.g., desktops).

For Ron, the real-time interaction is given top priority. Hence, to achieve such optimal execution in a mobile computing environment, the software components of the Google' glasses application should be computed in the most suitable location, considering their profiles. To efficiently distribute the application, we dissect it into class objects with consuming characteristics, such as processing time, energy consumption, and memory usage. This is the profiling step. For the modelling step, we use the *dependency graph* or call graph to represent the Google' glasses software, wherein each class may be a callee and/or caller vertex. Hence, a vertex represents a class, and an edge is an interaction between two classes. We valuate the vertices with the obtained statistics, such as the CPU-time, energy, and memory usage. For the edges, we use the call frequency between classes and data size exchange. The third step consists of partitioning the graph obtained from the second step. For this, we use a

well-known, min-cut/max-flow algorithm, such as Stoer Wagner. When the partitions are determined, code migration and result exchanges are carried out using one of the many communications models, including RPC, RMI, Service Oriented Architecture (SOA), Mobile Agents (MA) or streaming, (see Figure 2.1).

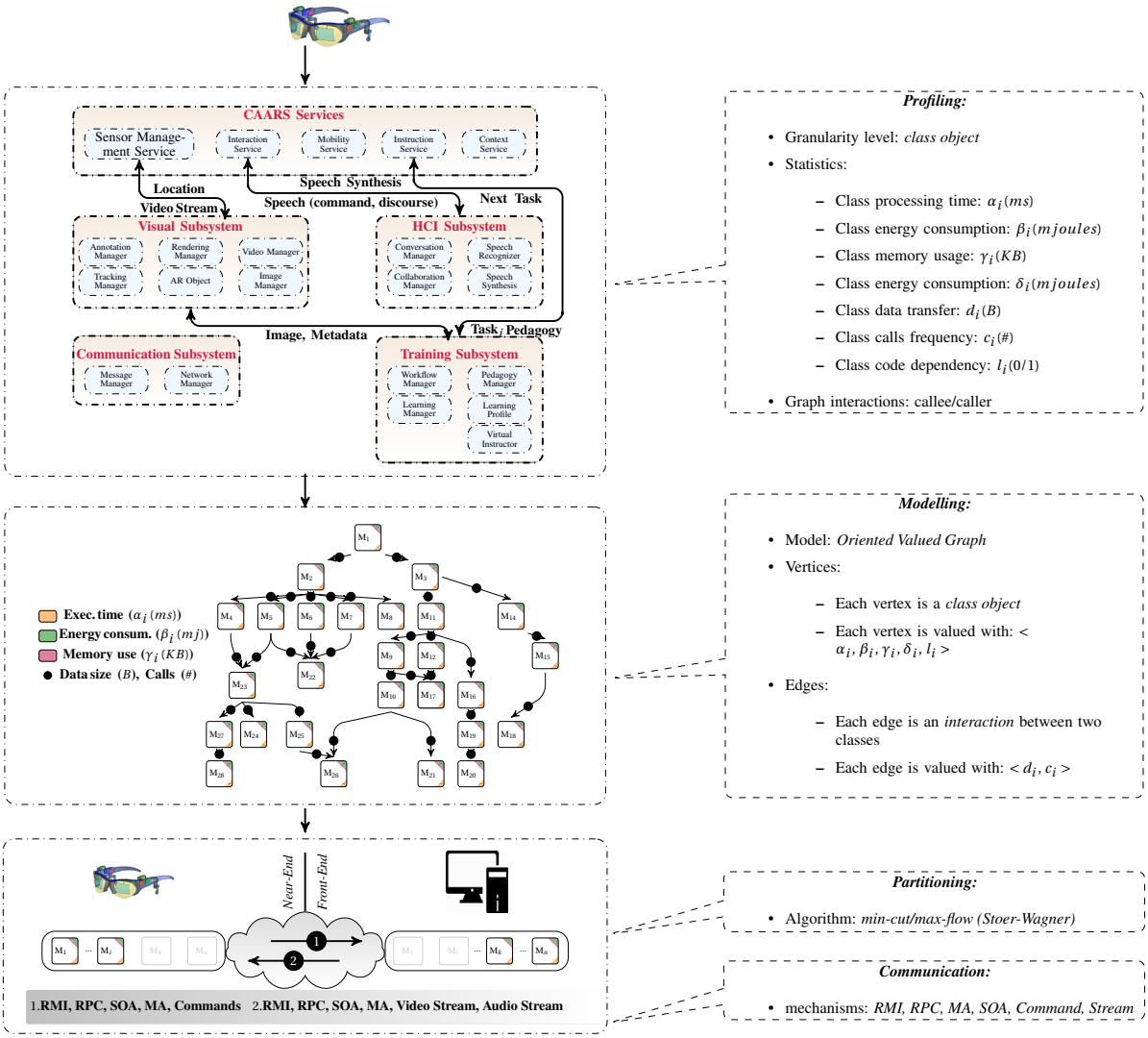


Figure 2.1: **Computation offloading process**

2.2.2 Offloading Types

When computation offloading follows the steps described above, it should confront with the performance and overhead[1] trade off. To this aim, three time-based approaches (static, dynamic, and hybrid) have been proposed in the literature and are discussed in the following.

[1]Local latency induced in the computation of the offloading steps.

Static offloading. As shown in Figure 2.2a, static offloading occurs at the start-time [32, 98, 166–168, 193, 213, 217, 237, 239, 277, 304]. In this mechanism, the application is partitioned only one time into two partitions: the local partition contains the local components, and the remote partition contains components being considered for remote execution. The partitioning process occurs prior to execution at the design or installation stage of the application, after analysing and profiling the code. The aim is to identify the software components with resource bottlenecks. These components will constitute the remote partition. Spectra [82], Chroma [18], and MISCO [67] are three frameworks using static offloading.

Dynamic offloading. Runtime or dynamic offloading is proposed to cope with the limitations of the static approach (which is becoming obsolete) with the dynamic processing load at runtime [36, 48, 59, 95, 109, 145, 208, 218, 255, 267, 291, 302, 310]. Indeed, the dynamic approach is more flexible, and it can adapt to different runtime conditions, such as changes in network latency, bandwidth, available energy, the computation loads on mobile devices and remote servers. The dynamic approach, as shown in Figure 2.2b, processes the aforementioned steps at run time. That is, all the offloading steps are done in real-time, when the application is running. MAUI [59] is a framework that is based on dynamic partitioning.

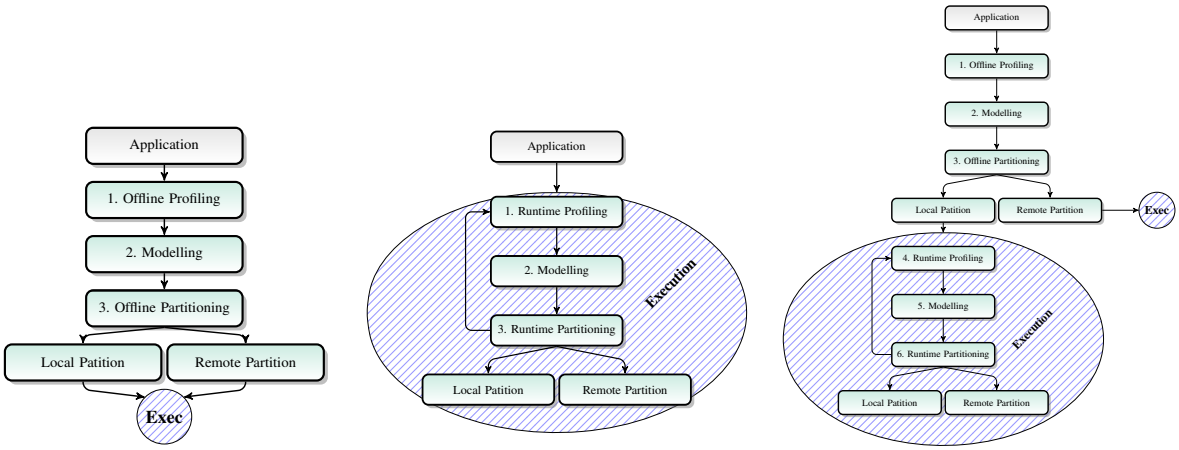
Hybrid offloading. In order to benefit from both static and dynamic offloading strengths, the hybrid (or *semi-dynamic*) approach has been proposed by Canepa and Lee et al. [32]. A part of the decision is done by the programmer-defined specification and static analysis tools, while the other part is done during the runtime as depicted in Figure 2.2c. The aim behind mixing the dynamic and static approaches was to minimize the side effects of profiling and waiting time (i.e., overhead). This solution does not always result in good performances as the execution time on a mobile device is short. Murarasu and Magedanz [198] have presented a middleware layer, between services and programs. This middleware supports static and dynamic reconfiguration of services and programs, monitors resource consumption, and manages the offloading to remote servers.

Comparison. Table 2.1 provides a comparison between the three time-based solutions.

2.2.3 Offloading Evolution and History

Computation offloading has been introduced over time in different forms with different objectives. Prior to 2000, researchers mostly focused on making computation offloading feasible, as the primary limitation in this period was network technologies. In the early 2000s, the main focus was on improving the gain of offloading by developing algorithms to decide *when* and *what* to offload. With the improvement in virtualization technology (such as, VMware[2], OpenStack[3], and Dockers[4]), network technologies (3G, 4G, and Wi-Fi with high throughput), and Cloud computing, computation offloading has moved in new

[2] `https://www.vmware.com/`
[3] `https://www.openstack.org/`
[4] `https://www.docker.com/what-docker`



(a) Static Offloading (b) Dynamic Offloading (c) Hybrid Offloading

Figure 2.2: **Time-based approaches for allocation decision**

Table 2.1: **Offloading type advantages versus disadvantages**

Type	Advantage	Disadvantage
Static	✓Simple, ✓Improve the performance, ✓No overhead, ✓No need for monitoring.	✗Not optimal, ✗Cannot guarantee the best partitioning, ✗Total knowledge of the program, ✗Does not support mobility, ✗Cannot adapt to changes (absence of monitoring), ✗Resource consumption are estimated in advance, ✗Extra efforts to integrate the offloading functionality.
Dynamic	✓*May* find the optimal solution, ✓Improve the performance, ✓Guarantee the best partitioning, ✓*May* support mobility, ✓Flexible and adaptive to dynamic environment, ✓Offloading functionality not considered in application dev.	✗Realization complexity, ✗Total knowledge of the program, ✗Increase the overhead, ✗Always delayed with the overhead elapsed time, ✗Waste local scarce resources (RAM and battery).
Hybrid	✓Reduce the overhead, ✓Improve the performance, ✓*May* support mobility, ✓Adaptive to available energy on the device.	✗Realization complexity, ✗Total knowledge of the program, ✗*Maybe* delayed with the overhead elapsed time, ✗Waste local scarce resources (RAM and battery),

directions. Since 2007, mobile users have embraced the concept of smartphones and tablets, which quickly became pervasive and ubiquitous devices. Thus, mobile developers have been motivated to create more and more mobile applications, including graphic rendering, face and speech recognition, mobile gaming, AR, and Virtual Reality (VR). These applications are CPU/GPU-intensive and energy consuming, which has resulted in more interest in and scope to computation offloading. Since 2015, computation offloading has gained even more ground, particularly with the emergence of MEC, FOG computing, and 5G, which allow real-time applications, such as interactive and gaming based-computation offloading.

Load Sharing. This work is considered the earliest research; it was first proposed in 1998 by Othman and Hailes [216] in order to conserve the resources of mobile devices. The authors were inspired by the *load balancing* concept in distributed systems. In this solution, the entire computation job is migrated to a remote server for computation. The system considers some metrics, including job size, available

bandwidth, and result size. The aim was to estimate whether the system can save energy or not. However, this method suffers from a lack of triangulation, wherein the jobs and data transit through a third-party device, which is responsible for finding the appropriate server and forwarding the results back to the mobile device. Furthermore, the computing servers are fixed, and the model does not take into account the user and code mobility at runtime.

Remote Execution. Remote execution became feasible with the work of Rudenko et al. [241, 242] in 1998. The authors addressed this solution for mobile computers (particularly *laptops*). This method saves energy when the remote processing cost is lower than the local processing cost. It is not sufficiently mature as it neglects the environment characteristics and the speed ration between the mobile computer and the remote server. Furthermore, the entire application needs to be migrated prior to execution.

Cyber Foraging. Cyber foraging is the process of dynamically augmenting the computing resources of a wireless mobile computer by exploiting wired hardware infrastructure. It is a re-definition of remote execution, introduced by Satyanarayana [248] in 2001 for pervasive computing. The two key ideas are the consideration of surrogates and dynamism in the remote execution process. Surrogates are static, idle computers located in the near vicinity of mobile devices that are acting as servers. These surrogates include personal computers and public servers; they are connected to uninterrupted power sources and wired Internet to provide unpaid computing resources. The entire application is stored in the mobile device. Therefore, a large overhead appears when identifying and partitioning intensive code, which can exceed conserved resources. Cyber foraging has been used in pervasive computing [17, 32, 82, 94, 124, 147, 275], grid computing [164, 212, 221, 274], and Cloud computing [31, 50, 51, 53, 59, 129, 130, 137, 151]. Cloudlet is a variant of cyber foraging introduced by Satyanarayana. The aim is to reduce network latency by moving the server closer to mobile devices [119, 120, 283, 303].

Computation Offloading. With the introduction of MCC, network communication technologies (3G, LTE, 5G, and Wi-Fi with high throughput), the latest efforts to address security in Cloud computing, the QoS, and reliability issues, computation offloading is gaining ground. Indeed, a significant amount of research has been performed on computation offloading, introducing various algorithms to make offloading decisions. These decisions are usually made by analysing parameters such as bandwidth, server speed, available memory, server load, network latency, and the amounts of data exchanged between servers and mobile systems. Offloading requires access to resourceful computers for short durations through networks, wired or wireless. These servers may use virtualization to provide offloading services so that different programs and their data are isolated and protected. Isolation and protection have motivated research on developing infrastructures for offloading at various granularity levels. Offloading may be performed at a method level, task level, application level, or Virtual Machine (VM) level. Chapter 3 is dedicated to computation offloading.

Edge-Based Computation Offloading. The last two years have characterized the emergence of Fog and MEC. Fog is an extension of Cloud computing, bringing network resources from the core network

to the edge network. It is gaining a lot of industry support, particularly from Cisco[5]. Fog is a highly virtualized platform, which provides a wide range of applications and services in highly distributed deployments, such as gaming, video streaming, augmented reality, and virtual reality. It offers low latency communication, which allows real-time delivery of data for delay-sensitive and health-care services. On the other hand, MEC has the same objective, but mainly targets 3^{rd} Generation Partnership Project (3GPP)-based mobile networks. MEC provides IT and computing capabilities within the Radio Access Network (RAN) in the close vicinity to mobile devices. It is heavily promoted by the European Telecommunications Standards Institute (ETSI), which is trying to develop a standard around MEC [73], defining a reference architecture and a set of Application Programming Interfaces (APIs)[6]. Combined with 5G mobile access, which aims to drastically reduce the end-to-end latency, MEC enables a plethora of novel mobile services that require short latency to access data or computation capabilities nearby, at the mobile edge. Among the envisioned services are computation-offloading-driven applications, which need to offload part of the execution of their applications code to a remote server. Chapter 6 is dedicated to MEC-oriented computation offloading.

We were interested in reviewing timely, both quantitatively and comprehensively, state-of-the-art research on the aforementioned transitions. We selected the *ACM Digital Library* and *IEEE Xplore Digital Library* to examine how many papers were published per year in the period 1998 to September 2017. The publications included journals, magazines, proceedings, and books. The aim of this research was to identify the key enabling concepts used in the landscape of computation offloading, to position our research on these axes, and to find the open research challenges to consider for our PhD thesis. The search keywords were *"load sharing," "remote execution," "cyber foraging," "computation offloading,"* and *"edge offloading."* Various papers used other words semantically similar to our keywords list. Therefore, we included in our search all papers wherein one of the following expressions appears: *"tasks offloading," "computation outsourcing," "MEC offloading,"* and *"FOG offloading."* We also looked for papers in which the keywords were switched in cases where the composition of the two words was semantically correct (e.g., we searched for papers using *"offloading task"* and *"outsourcing computation,"* but not *"offloading MEC"* or *"foraging cyber"*). The keywords were found in the title, abstract, research terms, or in the body of the papers. From the obtained results, we excluded the same papers according to the title.

Figure 2.3 shows the results. From 1998 to 2009, there were between 34 and 93 papers a year. Load sharing was the main research area during this period. Only a few papers were published on computation offloading during this era. We believe that computation offloading was limited by the lack of technological advancement in this epoch, especially in wireless technology, virtualization, and infrastructure. Since 2010, interest in computation offloading has increased dramatically, thanks to the arrival of new technologies, including materials (such as smartphones, Internet of Things (IoT), robots,

[5] https://developer.cisco.com/site/iox

[6] http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

and drones that involve the use of computation offloading), the variety of services and applications, and wireless technologies, such as Wi-Fi with high throughput, 3G, 4G, and LTE Advanced. We remind the reader that our search ended with *September* 2017; we are expecting more papers during the coming months.

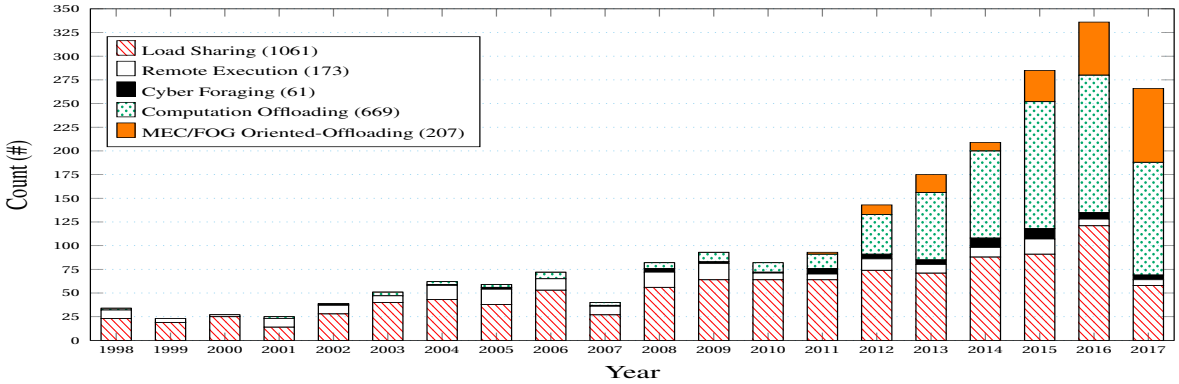


Figure 2.3: **Paper trend per step evolution**

We present in Table 2.2 the results of our search using a comparison of the above-cited approaches from the view of architecture used between the mobile device and the external platform, the cost and complexity of implementation, the QoE, the network delay, the possibility of migrating and partitioning the application, the mobility of the mobile user, and the security of data.

Table 2.2: **Features of offloading approaches**

Approach	Communication	Latency	Migration	Partitioning	Cost	Complexity	QoE	Mobility	Security
Load Sharing	Client-server (C/S)	High	Full job	No	High	Medium	Bad	No	NA
Remote Execution	C/S	High	Full, Partial	Static	Low	Low	Bad	No	NA
Cyber Foraging	C/S, P2P	Short	Full, Partial	Dynamic	High	High	Medium	No	No
Computation Offloading	C/S, P2P, AdHoc	Depends on server location	Full, Partial, VM Migration	Static, Dynamic, Hybrid	High	High	Good	Yes	Cloud
Edge-Oriented Offloading	C/S	Short	Full, Parital, VM Migration	Static, Dynamic	High	High	Good	Yes	Yes

Throughout this document, we use the terms computation offloading, remote execution, cyber foraging, and edge-based offloading interchangeably.

2.3 Metrics

Herein, we will be discussing the suitability of a range of metrics for an optimal offloading of mobile applications, particularly delay-sensitive applications. Figure 2.4 illustrates these metrics and classifies them into five areas namely: (i) *user preferences and requirements*, (ii) *mobile devices*, (iii) *application*

specifications, (iv) *external platform*, and (v) *network specifications*. We describe below the significance of each metric and how each impedes the offloading computation process.

1. User preferences and requirements. The offloading decision is affected by the user's physiological, physical, and mental states, his mobility, the goals to be achieved, the budget, the Quality of Service (QoS), the tasks and actions to process, and the individual and corporate preferences of the user [148]. Some users, unlike others, are more interested in confidentiality and the privacy level, and they do not use risky channels on the Internet to offload confidential data. Other users, are more interested in good QoS/QoE, considering their budget. Frequent mobility impacts the link failure across the mobile network [162], and therefore the computation offloading performance.

2. Mobile devices. The hardware configuration and OS used are important in computation offloading. Regarding the hardware, mobile devices have different configurations, which include CPU/GPU architecture and frequency, battery autonomy, memory capacity, and storage size. It is plainly evident that using a large memory prevents page fault exceptions and the paging process, a multi-core CPU better exploits parallel executions, and large battery autonomy increases the operational time for the mobile device. Considering the OS, each has a different architecture with different interruptions, scheduling policies, and algorithms for page replacement in caches, central memory management, and swaps. The access technology used is also vital in terms of the usability of computation offloading regarding latency and packet delivery, which can lessen the Quality of Experience (QoE).

3. Application specifications. Applications also impact the offloading decision. Indeed, code granularity, size, and data type are attributes impacting any decision. Moreover, the usual mobile applications exhibit six profiles, in terms of the users' needs, namely: CPU/GPU-, I/O-, memory- [318], network- [20], or security-intensive [33, 132], plus delay-sensitive [204]. Therefore, the application profile is another important attribute for decision making. In essence, computation offloading should consider the complexity of, data for, and nature of the application. However, some exceptions may occur; depending on the granularity level, some components of the application cannot be offloaded due to code dependency on: (i) the hardware, such as I/O and sensors; (ii) the OS (i.e., the component is a native code); or (iii) the user (user interfaces) [59, 94, 216, 217, 219].

4. External platforms. The remote infrastructure impacts highly computation offloading performance [322]. The server cost is measured with processor cycles, memory size, throughput, input data size, availability, elasticity, vulnerability to security attacks, and reliability in terms of delivering services within the agreed terms and conditions [129]. Moreover, the greater the distance between the server and mobile device, the higher the network latency and more modest the performance improvement.

5. Network specifications. According to the geographical scope and distance of connected nodes, we classify networks into three categories, namely, the Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN) [138]. Cloudlets are usually located in LAN networks, while the Cloud is hosted in the WAN [59, 250]. The necessity of mobility reduces the usability of

networks to those using only wireless technology (such as 3G/4G, and Wi-Fi). These technologies have different features (including latency, throughput, and packet delivery), which impact the computation offloading performance. Furthermore, the dynamic and rapidly changing mobile environment, user mobility, and weather conditions also play an important role in offloading attainment.

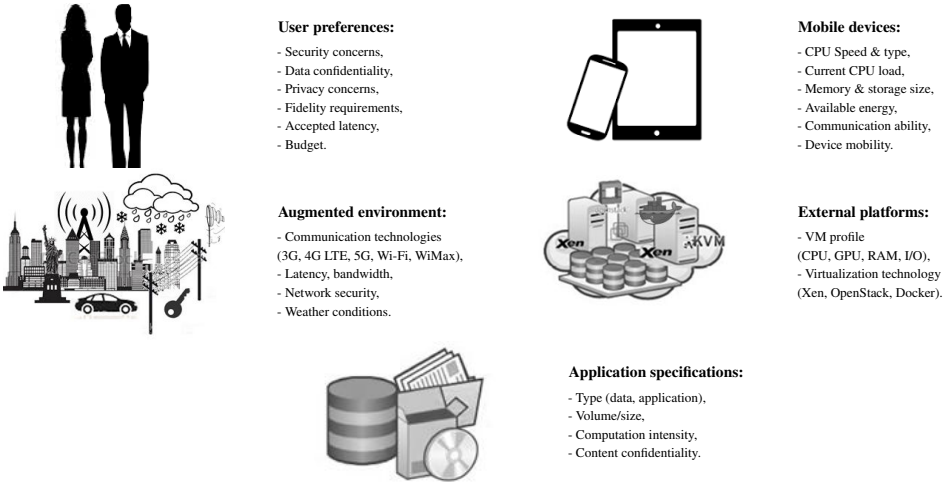


Figure 2.4: **Critical metrics influencing the offloading decision making**

2.4 Advantages and Disadvantages

Computation offloading is a critical technology that takes part in mobile Cloud computing, mobile edge computing, and 5G architecture. It is considered to be an enabler technology for intensive applications. In this section, we turn our attention to point out its merits and demerits.

2.4.1 Advantages

Computation offloading provides, but is not limited to, the following advantages:

1. Improves performance. By performance improvement, we mean execution time. Mobile users nowadays are interested in migrating their favourite applications and office work to their mobile devices. Nevertheless, these applications are usually computationally intensive, and either cannot run on powerless mobile devices, or can run with a low QoE. Due to computation offloading, running such applications on mobile devices became reality [48, 95, 193, 217, 237, 267, 277, 291].

2. Saves energy. Manufacturers are proposing multi-core ARM processors (CPU/GPU) with high clock speed that address power and cost requirements. Furthermore, movies, games, animations, and graphics have become an important part of mobile users' experiences. Increasing the performance of mobile devices means higher energy consumption. With computation offloading, energy can be saved by outsourcing the energy-intensive tasks from mobile devices [36, 59, 109, 145, 213, 239, 255, 304].

3. Augments storage. Due to the limited space on their devices, mobile users have to install and remove applications frequently and control the size and type of data stored on their devices. One solution for managing these data is to save them on an external storage-resourceful machine, such as servers on the Cloud. Such a solution maintains applications outside mobile devices, provides remote access to them, and updates the code without any I/O transactions, therefore enhance the QoE and saving energy [135, 136, 214, 215, 222].

4. Reduces memory usage. Recent operating systems are able to run various applications simultaneously. Some of these applications, such as social media (e.g., *WhatsApp* and *Facebook*), are memory-intensive applications. Keeping these running applications' states, will quickly fill the memory. Hopefully, computation offloading can reduce the memory utilization by offloading the memory-intensive parts of these applications [95].

5. Enriches the user experience. QoE and interactivity are the main criteria that impact the immersion of mobile users. According to Tolia et al. [279], an interaction delay higher than $1\,s$ is not acceptable to mobile users. A delay becomes tolerable within the range of $150\,ms$ and $1\,s$; below $150\,ms$, the mobile user's immersion is not affected. Computation offloading can guarantee a good QoE by focusing on the interaction delay.

6. Increases data safety. Using Cloud storage increases the size limit for and safety of stored data. Saving data on mobile devices is risky due to device robbery, physical damage, or malfunctions. It also degrades the QoE, and increases energy consumption, especially when a secure encryption is performed on these devices. By storing data on a secure Cloud [26, 292, 294, 309, 317, 328] users ensure data availability and safety anytime, anywhere, and from any mobile device.

7. Ubiquitous data access and content sharing. Adding to its temporal cost, copying data from one device to another is a risky practice as the data may be corrupted. Storing the data in the Cloud is a safe practice, and it improves the user experience. Indeed, remote storage enables mobile users to access their digital data regardless of time and place. Moreover, the data can be shared among different legitimate users.

8. Protects offloaded content. Cloud providers are deploying strict policies for security and privacy to protect user's code and data, ensure confidentiality, and secure properties and businesses. Virtualization in the Cloud also ensures privacy and security for customers. Furthermore, Cloud providers are using biometric security systems to protect their infrastructure and avoid unauthorized access; such systems include finger-, and retina-scan. Cryptography, frequent patching, and continuous virus signature updates are other forms of security [308].

2.4.2 Disadvantages

Computation offloading faces some limitations as outlined below.

1. Dependency on network conditions. The fact that computation offloading depends on network

communications is a major issue. Indeed, offloading gain is highly affected by network characteristics. Leveraging the Cloud infrastructure requires network communications, which is adversely affected by high WAN latency, non-guaranteed bandwidth, jitter, packet losses, and the non-deterministic traffic along the path.

2. Network and local latencies. Network latency affects significantly computation offloading since latency is a part of the interaction delay. Therefore, latency limits the use of computation offloading to delay-tolerant applications. Local latency is hit by the offloading steps, especially modelling and partitioning. For instance, graph partitioning is an NP-hard problem, and its resolution might take a long time. This leads to delayed decisions, which could be incorrect.

3. Code profiling. To make an offloading decision, the system needs to profile an application in order to estimate its resource consumption and obtain a call graph, which is an intensive operation. The resource consumption of the different software components (according to the granularity level) may include the processing time, energy consumption, and communications between modules. Profiling can provide incorrect estimations, fill up the storage, and considerably consume energy on mobile devices.

4. Tolerance for partitioning. Offloading frameworks should distinguish between *monolithic* and *distributable* applications [192]. Indeed, the application requesting offloading should tolerate component partitioning. Determining this is arduous, as a complete knowledge of the application is needed to identify code dependencies.

5. Framework complexity. When frameworks incorporate several parameters in decision making, the system becomes complex to manage and solve, which means the consumption of more energy and occupying more space due to the profiling of various metrics. Therefore, this increases the local latency and stresses the offloading process due to components moving back and forth between between local and remote partitions.

6. Security Issues. Leveraging the Cloud is a risky practice, especially for enterprise users, wherein competitors might access the offloaded confidential and financial data of other companies. Moreover, the risky Internet channel is a central concern. Indeed, packets might be captured and modified. Regarding mobile devices, security is also a challenge. In fact, malicious resource providers might attack mobile users to access to their private data or falsify offloading results.

2.5 Conclusion

Computation offloading is highly promoted for MCC, MEC, and the upcoming 5G systems. This will results in a plethora of applications and services. However, more investigation is necessary in order to evolve the computation offloading frameworks, optimize them, and move toward the next generation of frameworks with which to integrate with 5G scenarios, such as vehicles and m-heath, in which seamless and real-time constraints are on the top priority.

Throughout this chapter, we have defined the computation offloading concept. We surveyed the state-of-the-art computation offloading approaches from different perspectives. We highlighted the main steps employed in computation offloading architecture and projected them onto a scenario based on Google glasses. We classified the offloading mechanisms into three time-scale approaches, namely, static, dynamic, and semi-dynamic offloading. After that, we proposed an historical evolution of computation offloading that we quantified through the published papers in the literature. We classified the metrics impacting computation offloading into five areas (mobile devices, external platform, user requirements and preferences, applications specifications, and network characteristics). Finally, we discussed the advantages and disadvantages related to computation offloading. Overall, this introductory chapter provides an understanding of the computation offloading concept. In the following chapters, an exhaustive taxonomy of computation offloading will be provided.

CHAPTER 3

COMPUTATION OFFLOADING: TAXONOMY REVIEW

3.1 Introduction

Computation offloading is considered to be corner stone of MCC, mobile gaming, MEC, and the upcoming 5G networks. Computation offloading allows code distributing and portability, and improved performance. However, deploying such a system is complex due to a large number of problems, which are, but are not limited to, network and Cloud heterogeneity, application complexity, and code ability for offloading. Researchers in industry and academia have devised many frameworks for computation offloading, the latest of which manage some challenges and limitations of past frameworks. These research works have covered the actors involved in computation offloading, as defined in Chapter 2, namely, the mobile user, mobile device, application, network, remote infrastructure, and offloading framework, each of which describes a set of challenges and issues. We propose in the following, a comprehensive survey that studies state-of-the-art computation offloading from different points of view according to the actors that impede the offloading process. We propose a thematic taxonomy that reviews comprehensively and qualitatively, computation offloading approaches from different perspectives. That is, from each side, we identify several challenging concepts described in past studies. Finally, we review several frameworks and propose a qualitative comparison from different perspectives.

3.2 Computation Offloading Taxonomy: From the User Perspective

This section provides the reader with state-of-the-art, optimization-based strategies for frameworks. A comparison between various frameworks on the basis of significant optimization parameters that affect performance is also provided in this section. Some mobility models described in the literature are highlighted, after which we discuss the mobile users' concerns regarding computation offloading. Figure 3.1 depicts the axes on which we focus in this section, namely the objective function, mobility,

and concerns regarding mobile users in a computation offloading environment.

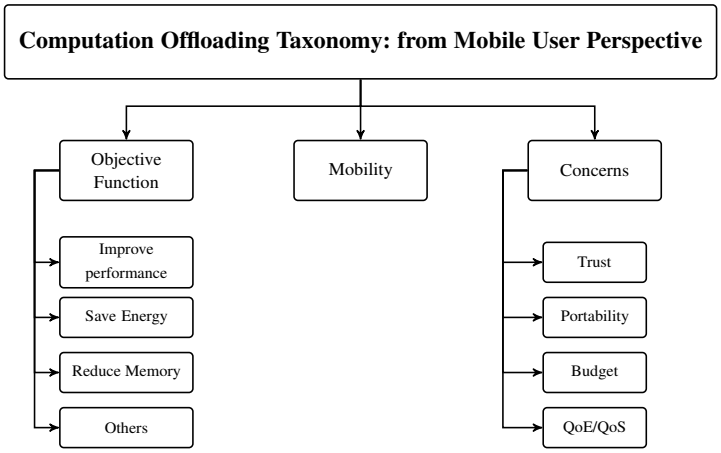


Figure 3.1: **User-based computation offloading taxonomy**

3.2.1 Objective Function

The aim behind computation offloading is to fulfil the requirements of various types of applications by leveraging remote infrastructures. Most frameworks focus on three main requirements: (i) *performance*, which is expressed by execution time, (ii) *energy* expressed by the energy saved through computation offloading, and (iii) *memory*, where the objective is to save or extend the memory of the mobile device. We study in Appendix A.1, the gains obtained by computation offloading through proposed mathematical models. Hereafter, we classify state-of-the-art computation offloading frameworks according to the number of involved parameters in the objective function. This results in three models, namely, the *mono-*, *bi-*, and *multi-* objective models.

3.2.1.1 Mono-objective model

The mono-objective frameworks mainly focus on the optimization of a single parameter. It is a straightforward optimization model. However it cannot meet the requirements of applications in an heterogeneous execution environment. Herein, we present some frameworks that use mono-objective models:

(a) CloneCloud [50] is a Cloud-based system that clones the mobile device platform with a VM-hosted platform in the Cloud. The implementation is based on *Java DalvikVM*. It is proposed to empower mobile applications through offloading computation to the Cloud. It is a dynamic offloading framework, with the objective function to optimize the overall execution cost of mobile applications. The authors have split the execution cost into a computation cost, $Comp(E)$, and a migration cost, $Migr(E)$. The

energy consumption cost is is expressed as:

$$C(E) = Comp(E) + Migr(E) \tag{3.1}$$

$$Comp(E) = \sum_{i \in E,m} [(1 - L(m).I(i,m).C_c(i,0) \quad + \quad L(m).I(i,m).C(i,1))] \tag{3.2}$$

$$Migr(E) = \sum_{i \in E,m} R(m).I(i,m).C_s(i) \tag{3.3}$$

For every invocation $i \in E$, the computation cost $Comp(E)$ takes its value from the mobile device cost variables $C_c(i,0)$ or the clone cost variables $C_c(i,1)$ depending on whether the invoked method m will be run on the mobile device or in the Cloud. The migration cost $Migr(E)$ sums the individual migration costs $C_s(i)$ of each invoked method i that is a migration point. The optimization problem has been modelled using an ILP wherein the objective is to find value of a decision variable $R(i)$ for the method i that minimizes the $\sum_{E \in S} C(E)$, under the constraints:

$$L(m_1) \neq L(m_2) \quad \forall m_1, m_2 : \quad DC(m_1, m_2) = 1 \quad \wedge \quad R(m_2) = 1 \tag{3.4}$$

$$L(m) = 0 \quad \forall m \in V_m \tag{3.5}$$

$$L(m_1) = L(m_2) \quad \forall m_1, m_2, C : \quad m_1, m_2 \in V_{Nat_c} \tag{3.6}$$

$$R(m_2) = 0, \quad \forall m_1, m_2 : \quad TC(m_1, m_2) = 1 \wedge R(m_1) = 1 \tag{3.7}$$

Equation 3.4 is a soundness constraint, which stipulates that a method m_1, which causes a migration to happen, cannot be located in the same location as its caller method, m_2. Constraint 3.5 is a dependency constraint, wherein methods annotated to be pinned on the mobile device run only on the mobile device. Constraint 3.6 ensures that methods depending on the same class state are collocated, at either location. Finally, constraint 3.7 ensures that all methods transitively called by a migrated method cannot, themselves, be migrated.

As stated above, $R(m)$ is the decision variable that represents the modification (i.e., $R(m) = 1$) or not (i.e., $R(m) = 0$) of the method m, which is done through an insertion of a migration point in the entry of the method.

(b) CODM [287] is a middleware aimed at selecting and deploying the application configuration that offers the best quality possible, according to the current available resources and connectivity. The deployment granularity of the middleware is a bundle. For each bundle, the framework provides multiple configurations with different levels of resource consumption and quality levels. The objective function is given by:

$$Max_{configurations} \sum_i w_i$$

$$s.t. \begin{cases} \forall m : \ \sum_i X_{im} \times w_i \leq M_m \\ \forall i : \ \sum_i X_{im} = 1 \end{cases} \tag{3.8}$$

where w_i represents the cost (CPU power) of the i^{th} bundle, and X_{im} is the decision variable that is equal to 1 if the bundle i is assigned to the machine m ($m \in 0,1$) (mobile device or Cloud) and 0 otherwise. The first constraint ensures that the sum of the weights of all bundles deployed on the machine m cannot exceed the maximum allowed weight, while the second constraint stipulates that a bundle i of the configuration is deployed either on the mobile device or on the Cloud. The framework adapts dynamically the configuration and deployment of the bundles, but also degrades gracefully the quality level in some cases. However, the framework requires different configurations of the application and relies on the OSGi framework.

(c) MAUI [59] is an energy-aware framework that operates on method granularity. It employs both static and dynamic partitioning. The framework requires programmer efforts to annotate the application methods as remotable or not. MAUI uses a timeout mechanism to detect connection failures with the remote server. The authors have drawn the optimization problem for an ILP as follows:

$$\begin{aligned} & Max \sum_{v \in V} I_v \times E_v^l - \sum_{(u,v) \in E} |I_u - I_v| \times C_{u,v} \\ & s.t. \sum_{v \in V} \left((1 - I_v) \times T_v^l + \left(I_v \times T_v^r \right) \right) + \sum_{(u,v) \in E} \left(|I_u - I_v| \times B_{u,v} \right) \leq L \ \ and \ \ I_v \leq r_v, \ \ \forall v \in V \end{aligned} \tag{3.9}$$

where I_v is the decision variable, and $I_v = 0$ ($I_v = 1$, resp.) if the method v is executed locally (offloaded, resp.). E_v^l and T_v^l denote, respectively, the energy consumption and execution time of method v locally. $B_{u,v}$ represents the necessary program state when u calls v, and $C_{u,v}$ is the energy cost of the transferring state, and finally, r_v indicates if the method v is remotable (i.e., $r_v = 1$) or not remotable (i.e., $r_v = 0$).

(d) Yang et al. [312] have studied how to optimize the computation partitioning of a data stream application between mobile devices and the Cloud to achieve a maximum speed/throughput in processing the streaming data. The partitioning problem has been formulated as a weighted dataflow graph and then drawn into an ILP with the objective to maximize the data stream throughput. The ILP is given by:

$$\begin{aligned} & Max_{x_i, y_{i,j}} \ \ TP = \frac{1}{t_p}, \ \ i, j \in \{0, 1, ..., v+1\} \ \ where \\ & t_p = max \left\{ max_{i \in V} \left(x_i \times \frac{S_i}{\eta \rho} \sum_{i \in V} x_i \right), max_{(i,j) \in E} \left(\frac{d_{i,j} \left(x_i, y_i \right)^2}{y_{i,j}} \right) \right\} \\ & s.t. \begin{cases} \sum_{i,j \in E} y_{i,j} (x_i - y_i)^2 = B, \\ y_{i,j} > 0, \\ x_0 = 1, \\ x_{v+1} = 1, \\ x_i = 0 \, or \, 1, \ \ i \in \{1, 2, ..., v\} \end{cases} \end{aligned} \tag{3.10}$$

ρ and η represent, respectively, the CPU capability of the mobile device and its workload, and B is the current network bandwidth. Next, x_i and $y_{i,j}$ are the core variables, and x_i is the decision variable:

when $x_i = 1$ ($x_i = 0$, resp.), component i is computed locally (offloaded, resp.). Finally, $y_{i,j}$ is the wireless bandwidth allocated to channel (i, j).

3.2.1.2 Bi-objective model

In bi-objective models, the main focus is on optimizing two parameters. Bi-objective optimization-based frameworks are relatively complex compared to the mono-objective frameworks. These frameworks aim to optimize simultaneously the two parameters in the objective function. This produces a better fit to diverse environments and fulfilment of the applications' requirements.

(a) AIOLOS [285] is a mobile, middleware-based cyber foraging system. The framework operates at method granularity level. It relies on an estimation model of both local and remote resources, with a network state to dynamically decide whether or not a method should be offloaded. AIOLOS is built on OSGi[1], with the objective to optimize both the local execution time and energy consumption. The authors use a history-based profile to estimate the local execution time $\hat{T}_{CPU,local}$. The formula to estimate the remote execution time is given by:

$$\hat{T}_{remote} = \frac{1}{\alpha} \times \sum_{i \in RM} (\hat{T}_{CPU,local_i}) + \frac{1}{\beta} \times (A + R) + \gamma + \sum_{j \in CM} \left(\hat{T}_{CPU,local_j} + \frac{1}{\beta} \times (A_j + R_j) + \gamma \right) \tag{3.11}$$

where α is the speedup factor of processing, and β and γ represent, respectively, the network bandwidth and latency. RM and CM define, respectively, the collection of remotely executed methods and the collection of callbacks. Finally, A and R are, respectively, the input and output sizes. For energy optimization, the authors have presented a simple decision model in which a method is offloaded only when the energy consumed by sending and receiving bytes to and from the server is smaller than the energy saved by offloading the computation. The proposed formula is given in Eq. 3.12, where E_{CPU} denotes the energy consumption per time unit by the CPU, and E_{TR} and E_{RCV} denote, respectively, the energy cost for the transmission and reception of a byte of data.

$$\hat{E}_{saved} = E_{CPU} \times \sum_{i \in RM} (\hat{T}_{CPU,local_i}) - E_{TR} \times A - E_{RCV} \times R - \sum_{j \in CM} \left(E_{RCV} \times A_j + E_{TR} \times R_j \right) > 0 \tag{3.12}$$

3.2.1.3 Multi-objective model

The multi-objective optimization-based frameworks focus on the optimization of multiple parameters. These frameworks are more complex in comparison to the mono-objective and bi-objective frameworks. The multi-objective frameworks include multiple parameters in the objective function. These frameworks are more adapted to heterogeneous computing environments. Hereafter, we present some of these frameworks.

(a) AIDE [95] is a fine-grained dynamic offloading system. The authors have focused on three metrics: *bandwidth requirement, interaction frequency*, and *memory size*. A coalescing process based on a minimum-cut heuristic algorithm [273] has been implemented to find all possible 2-way cuts of

[1] A module system and service platform enabling runtime deployment of application components, called bundles, implemented in Java.

the execution graph. In the beginning, the authors created two partitions. The first partition contains the classes that should be executed on the mobile device, these nodes are merged together to make one node. The second partition encloses the rest of the nodes. Next, the authors have repeated the following rules, until all the nodes are merged together in the first partition: (i) They selected the least memory-intensive class with the largest interaction frequencies and bandwidth requirements from the second partition in such a way that this node is a successor node to the merged node (partition one); (ii) next, they merged the selected node with partition one; (iii) in each step, a possible 2-way cuts is generated, and (iv) at the end of the process, the authors selected the 2-way cut that minimizes the metrics defined above. Therefore, to select the best 2-way partitioning, the authors have to compare between two metrics C_k, C_l as follows: $C_k > C_l$ if and only if:

$$w_1 \times \frac{b_{i,k} - b_{i,l}}{b^{max}} + w_2 \times \frac{f_{i,k} - f_{i,l}}{f^{max}} + w_3 \times \frac{M_l - M_k}{M^{max}} > 0 \tag{3.13}$$

where $w_i (1 < i < 3)$ is the weight of the i^{th} metric in the decision making; b^{max}, f^{max} and M^{max} represent the maximum values of the inter-class bandwidth requirement, inter-class interaction frequency, and class memory size, respectively; $b_{i,j}$, $f_{i,j}$, and M_i represent, respectively, the bandwidth requirement from class i to class j, the number of interactions between the two classes i and j, and the memory size of class i.

(b) Elastic Application [322]. Zhang et al. have designed a framework for elastic applications enabling the use of Cloud resources in an optimal and transparent manner. The proposed framework includes four attributes of the optimization-based model; energy, monetary cost, throughput, and security/privacy. The authors, through their model, wish to minimize both energy consumption and the cost, while maximizing the throughput, and security and privacy. Applications are partitioned into various *weblets* that are replicated across multiple Clouds aiming at improving the availability and reliability. The objective function is given by:

$$y^* = arg\,max_y\, p(y) \prod_{i=1}^{L} p(x_i|y) \prod_{j=1}^{M} p(z_j|y) \tag{3.14}$$

where x and z are vectors; x contains values of different device status components, including the throughput, memory usage, upload bandwidth, and file cache; z contains a user's preferred option values, such as processing speed and monetary cost; y represents the sum of all configurations; and L (M, resp.) is the number of components in the status (preference, resp.) vector. As the framework is for multi-objective decision making, additional computing resources are needed to solve the optimization problem, which may take a long time.

(c) Mobile Augmentation Cloud Services (MACS) [144] is a middleware enabling a lightweight partitioning with seamless execution of applications in the Cloud. The framework is devised for android applications. To make a decision to offload a service or not, the framework first monitors the resources at both locations, then creates an optimization problem that combines memory, transfer, and execution

time costs. The cost function is represented as follows:

$$Min_{x\in 0,1}\left(C_{transfer}\times w_{tr}+C_{memory}\times W_{mem}+C_{CPU}\times W_{CPU}\right) \tag{3.15}$$

Where:

$$\begin{cases} C_{transfer}=\sum_{i=1}^{n} code_i\times x_i+\sum_{i=1}^{n}\sum_{j=1}^{k} tr_j\times\left(x_j\,XOR\,x_i\right)\\ C_{memory}=\sum_{i=1}^{n} mem_i\times(1-x_i)\\ C_{CPU}=\sum_{i=1}^{n} code_i\times a\times(1-x_i)\end{cases} \tag{3.16}$$

In the above equation, n is the sum of the offloaded modules; $code_i$, mem_i, and tr_i represent, respectively, the code size, memory cost, and transfer cost of module i; and x_i is the decision variable, which represents the decision made for each module in term of whether it should be offloaded (i.e., $x_i = 1$) or computed locally (i.e., $x_i = 0$). Even though MACS is a lightweight and dynamic framework, it requires developer efforts to structure the application program into a model. Furthermore, the offloading performance is impacted by the profiling and partitioning steps.

(d) Adaptive (k+1) [217]. Ou et al. have proposed an adaptive, multi-constraint partitioning algorithm for offloading in pervasive systems. The framework partitions an application into k offloadable partitions and one unffloadable partition. The partitioning algorithm is applied to a dynamic multi-cost graph. The vertices correspond to the application classes, each valued with an n-tuple, $< w_1, w_2, ..., w_n >$, $(n \neq 0)$ costs such as memory, bandwidth, and execution time. The edges represent the interactions among classes. The authors have used a heavy-edge and light-vertex matching algorithm to coarsen the multi-cost graph. Hence, a composite-vertex-weight is used to replace the weight vector. The composite vertex weight is computed as: $w^v_{composite} = \sum_i \varepsilon w^v_i$, where $i = 1, 2, ..., n$, n is the length of the weight vector, and ε is the importance of the i^{th} weight in the vector, which is assigned based on the scarcity of the corresponding resource. The higher the scarcity of a resource, the higher ε will be. The authors have formulated the problem as follows: Given an application graph $G(V, E)$ and a non-negative integer k^l, the adaptive (k+1) multi-constraint partitioning algorithm has to find one local (unoffloadable) partition V^U and k disjoint remote (offloadable) partitions $V^O_1, V^O_2, ..., V^O_k$, which should satisfy the following constraints:

$$\begin{cases} \cup_{i=1}^{k} V^o_i = V - V^U \;\; and \;\; V^o_i \cap V^o_j = \emptyset \;\; for \;\; 1 \le i, j \le k \;\; and \;\; i \neq j\\ The \;\; edge-cut \;\; of \;\; \forall V_i, V_j \in V^U, V^o_1, V^o_2, ..., V^o_k \;\; C^{(i,j)} = \sum_{(u,v)\in E, u\in V_i, v\in V_j, u\neq v}\\ \forall i, j : \psi^i_j \le (T^i_j \pm \delta^i_j)\end{cases} \tag{3.17}$$

ψ^i_j is the sum of the j^{th} vertex-weight in partition V_i: $\psi^i_j = \sum_{v\in V_i} w^v_j$; T^i_j and δ^i_j are the multiple constraints representing threshold and partitioning fluctuating factor in partition V_i according to the j^{th} vertex weight. These two parameters define, respectively, the lower and upper bound of the constraints. When a composite vertex weight is used, ψ^i_j becames $\psi^i_{composite}$, and it is computed by $\psi^i_{composite} = \sum_{v\in V_i} w^v_j$.

Table 3.1 presents the comparative summary of aforementioned models considering various metrics. ●, ■, and ▲ mean, respectively, low, medium, and high consumptions. ✓ and ✗ show whether the functionality is supported or not.

Table 3.1: **Comparison between mono-, bi-, and multi-objective optimization-based frameworks**

Framework	Type	Optimization Objectives	Latency	QoS Support	Overhead	Scalability	Annotations	Energy Consmp.	Operational Cost
CloneCloud [50]	Mono-Objective	Execution Cost	●	NA	▲	✗	✗	●	▲
CODM [287]	Mono-Objective	Optimize Deployment	●	✓	●	✗	✓	●	●
MAUI [59]	Mono-Objective	Optimize Energy Consmp.	●	✗	▲	✗	✓	●	●
Yang et al. [312]	Mono-Objective	Optimize Throughput	✗	✗	■	✓	✓	●	●
AIOLOS [285]	Bi-Objective	.Reduce Latency, .Minimize Energy	●	NA	NA	✓	NA	●	■
AIDE [95]	Multi-Objective	.Min. Bandwidth, .Min. Network Communication, .Min. Memory	●	NA	●	NA	✓	NA	●
Elastic Application [322]	Multi-Objective	.Min. Energy, .Max. Throughput, .Min. Monetary Cost	▲	✓	▲	✓	✓	●	▲
MACS [144]	Multi-Objective	.Min. Energy, .Max. Security, .Min. Exec. Cost	●	✗	▲	✗	✓	●	●
Adaptive (k+1) [217]	Multi-Objective	.Min. Bandwidth, .Min. Memory, .Min. Exec. Cost	NA	✗	▲	✓	✓	NA	●

3.2.2 Mobility

As far as we know, mobility has not been fully studied in computation offloading. Only a few studies have addressed mobility when performing computation offloading.

Mob-aware [158] is a mobility-aware offloading decision maker, which considers future network changes based on user mobility. The Mob-aware gathers previous user movements and network changes, then builds a mobility model to predict future network changes. The authors have used Markov theory to draw the mobility model and calculate the probability of moving to a certain Wi-Fi Access Point (AP). Each state s_i in the model represents a Wi-Fi AP characterized with a bandwidth bw_i and a staying time st_i, while a transition represents the probability of visiting a certain state. Using this model, Mob-aware estimates the expected remote and local execution time. Mob-aware explores each possible transition to estimate the response time based on the bandwidth bw_i, and the staying time in each state st_i. Then, the engine calculates the probability of taking a possible path (i.e., a plausible sequence of transitions). The probability $P(Q_i)$ of taking a path Q_i is given by:

$$P(Q_i) = P(s_0, s_1, ..., s_n) = \prod_{k=2}^{n} P(s_k | s_{k-1}, s_{k-2}) \tag{3.18}$$

Mob-aware expects the response time R of remote execution using equation 3.19, where R_i is the response time of remote execution on a path Q_i. The throughput on each path is expected using bandwidth bw and the state time on each state st. Also, d_u and d_r are the input and result data size,

respectively.

$$R = \sum_{i} P(Q_i) \times R_i(Q_i, d_u, d_r) \tag{3.19}$$

Ou et al. [218] proposed an analytic model for computation offloading in a mobile wireless environment using the mobility Random Waypoint (RWP) scheme [122]. The authors considered a one-dimensional mobility. They defined a continuous random variable, X, to express the user mobility coordinate, assuming a random speed $v_i \in [v_{min}, v_{max}]$. The asymptotically stationary probability density function (p.d.f) of location X in $[0, d]$ considering the RWP model with a non-zero speed mobility is approximated by: $f_x(x) = -\frac{6}{d^3}x^2 + \frac{6}{d^2}x, \;\; 0 < x < d$ A surrogate is not reachable when the difference between the mobile user's location, x^m, and the surrogate's position, x^s, is greater than the surrogate's radio transmission radius, D, (i.e., $|x^m - x^s| > D$). Since surrogates are immobile, their radio coverage can be obtained. If m denotes the number of surrogates on segment $[0, d]$, therefore there are $m + 1$ sub-segments $([0, x_1^s - D), (x_1^s + D, x_2^s - D), ..., (x_m^s + D, d])$, which are out of the radio coverage. The probability of surrogate being un-reachable at anytime is given by:

$$\alpha = \int_0^{x_1^s - D} f(x)dx + \sum_{i=1}^{m-1} \int_{x_i^s + D}^{x_{i+1}^s - D} f(x)dx + \int_{x_m^s + R}^{d} f(x)dx \tag{3.20}$$

Events of unreachable surrogates are modelled with a Poisson process. P is a random variable representing the time to the first epoch in which the surrogates are unreachable. P follows an exponential distribution with parameter α.

$$f_p(P) = \alpha e^{-\alpha P} \;\; , P \geq 0 \tag{3.21}$$

The authors have modelled the running application with four states; *non-offloading execution (S_{NE})*, *offloading (S_{OL})*, *offloading execution (S_{OE})*, and *failure and recovery (S_{FR})*. Offloading events follow a Poisson process with a rate β. During the offloading and failure events, periods of time M and R are needed to, respectively, migrate data to a remote surrogate and to recover after failure. M and R are random variables following a general distribution. When a failure occurs, application execution is interrupted next, the failure is recovered, and the execution is restarted from the scratch in the same period as the last execution. The failure occurs during the states S_{OL} and S_{OE} due to the surrogate's unreachability when the user is moving, and in state S_{FR} when the failure recovery aborts. In these cases, α becomes the probability of failure.

The authors have expressed the execution time without failure using the following Equation 3.22:

$$T_{non-failure}(n) = \left(1 - \frac{(\omega - 1)\sigma}{\omega}\right) T'(n) \tag{3.22}$$

where n is the number of tasks composing the application, ω is the speedup factor of the surrogate compared to the mobile device, σ is the proportion of sub-tasks performed by the surrogate, and $T'(n)$ is the local execution time of the application on the mobile device. The execution time with failure is

given by Equation 3.23.

$$T^*(n) = \begin{cases} T(n) + \sum_{i=1}^{h} M_i & if\ P \geq T(n) + \sum_{i=1}^{h} M_i \\ P + R^* + \hat{T}^*(n) & if\ P < T(n) + \sum_{i=1}^{h} M_i \end{cases} \tag{3.23}$$

where h is the number of transitions from the state S_{NE} to the state S_{OE}, R^* is the failure recovery time in the presence of failures within the recovery period R, and P is the time to the first failure after starting recovery.

- if $P \geq T(n) + \sum_{i=1}^{h} M_i$ then, h offloading operations are made before finishing the n tasks without failure. Therefore, the offloading time is given by $\sum_{i=1}^{h} M_i$, and the total execution time is expressed by $T(n) + \sum_{i=1}^{h} M_i$. The random variables M_i, $i = 1, 2, ..., h$ are independent and identically distributed (i.i.d).
- if $P < T(n) + \sum_{i=1}^{h} M_i$ then a failure will occur before finishing the n tasks and making h offloading operations. Therefore, a nested failure occurs, and a new recovery operation is repeated after P, with a recovery time R^*.

MOSys [123] is a seamless computation offloading framework. MOSys uses the Software-Defined Networking (SDN) [313] paradigm to manage user mobility and caching techniques to reduce response time. The framework MOSys is composed of three entities; a *middleware client* that is responsible for the collection of data, such as application type, network, and Cloud parameters; a *network controller* that manages user mobility, tracks his location, and maintains his IP address; and a *middleware server* that performs the offloading requests.

The architecture manages two types of interactions: (i) between the mobile device and the OpenFlow controller [2] to manage mobility, discover Cloud resources and services, and perform offloading computation steps; and (ii) between the OpenFlow controller and the Cloud, which accepts or rejects computing the offloaded tasks, monitors the Cloud resources, and performs caching. The middleware is composed of classical offloading components, such as profiler, decision engine, and offloading manager, and a remote caching component to store data in a cache for future use. The network controller is composed of: (i) OFMAG, responsible for user's mobility and managing handovers and connectivity; and (ii) OFLMA creates OpenFlow rules to manage the traffic and maintain data to identify handovers. The authors used the Proxy Mobile IPv6 (PMIPv6) [97] to devise these two components.

3.2.3 Challenges

- *Trust*: Establishing trust between mobile end-users and Cloud providers is central in MCC. Several studies have been conducted in this axis to build trust in Cloud resources [197, 246, 297]. However, risky Internet channels and Cloud infrastructure heterogeneity impact the trust between mobile

[2] http://archive.openflow.org/

users and Cloud providers. Encryption/decryption, authorization, and authentication techniques are used in the MCC to overcome the trust challenge. However, these techniques increase the communication overhead and resource consumption on mobile devices.

- *Data Portability*: Cloud providers have alleviated the locked-in problem of mobile device users to facilitate data migration through non-uniform mobile devices. Yet, portability of data is still a challenge for Cloud consumers. For instance mobile users are unable to transfer data from Android-based to iOS-based mobile devices. On top of that, Cloud service heterogeneity causes a provider lock-in problem. Provider lock-in is a real concern for customers and an attractive issue in business [182]. Solutions, such as the proposed middleware [24] and standardization of Open Cloud Computing Interface (OCCI)[3] are some approaches addressing the portability issues in MCC.
- *Budget*: In order to use the Cloud for computation offloading, the costumer should pay for the services offered, which include resource reservations, QoS, reliability, and security [245], according to the negotiated Service-Level Agreement (SLA) with the Cloud provider. Moreover, to offload computation, the user should have access to the internet through wired or wireless connectivity. When using radio communication, the user has generally a limited size of data to upload and download; if this quota is exceeded, the user will have to pay for additional data, usually at a very expensive rate.
- *QoS/QoE*: Concerns in this area are interaction delay, immersion, and in the case of multimedia, the number of frames generated per second. The following chapters discuss these concerns in more details.

3.3 Computation Offloading Taxonomy: From the Mobile Device Perspective

In this section, we initiate the reader to mobile device's limitations and characteristics that have pushed for the need to use computation offloading. We are particularly interested in mobile device platforms that have been used for computation offloading. We also want to describe mobile devices from the hardware capabilities view to understand their limitations in term of resources. Finally, wireless communication also matters in computation offloading. Indeed, each technology has its characteristics, including throughput, energy consumption, latency, supported services, which highly impact computation offloading. Figure 3.2 highlights these interests.

3.3.1 Platform

A platform is the operating system (OS) layer that make mobile devices operational, and the operating system manages user applications. The manufacturers of mobile devices can be separated, according to the OS running on these devices, into *light OS* and *full OS*.

[3] http://occi-wg.org/

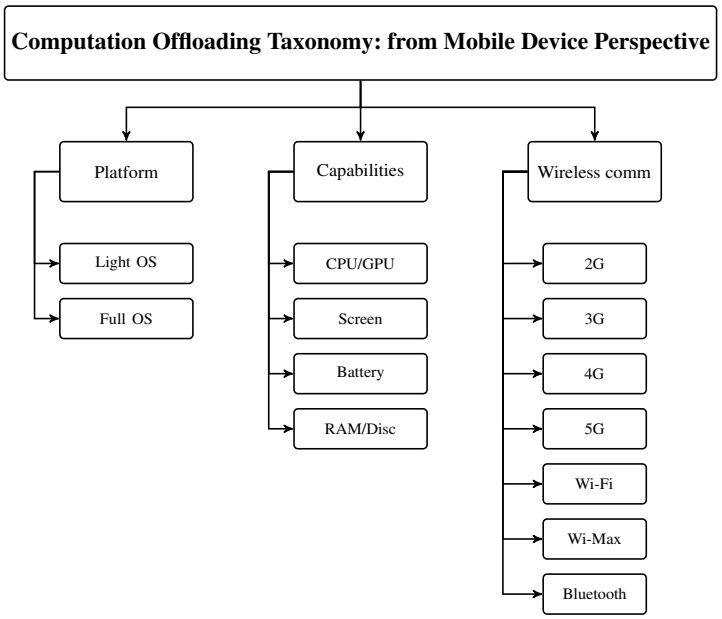


Figure 3.2: **Mobile device-based computation offloading taxonomy**

- *Light OS* concerns smartphones and tablets, which include Android, iOS, and Symbian.
- *Full OS* is supported by laptops and some surfaces. It includes, Windows (e.g., 7/8/10), Linux (e.g., Mandriva, RedHat, and Debian), and MAC OS.

3.3.2 Wireless Communication

To communicate with the Cloud/Edge environment that is computing the offloaded tasks, mobile devices are using wireless communication such as cellular networks (e.g., 2/3/4/5G, and Wi-Max) or Wi-Fi access point. A comparison study of wireless protocols: Bluetooth, ultra-wideband (UWB), ZigBee, and Wi-Fi was proposed in [156]. The authors have evaluated the protocols' main features and behaviors in terms of various metrics, including the transmission time, data coding efficiency, complexity, and power consumption. In [65], a comparison between WiMAX, HSPA and LTE access networks was provided regarding the power consumption. In the computation offloading arena, several works have been proposed based on different wireless communication technologies. In [139], authors have tested their framework, ThinkAir, for N-Queen-Puzzle, Face detection, and virus scanning for both 3G and Wi-Fi connectivity. In [323], the authors have proposed and experimented a method for refactoring Android Java code for on-demand computation offloading in 3G networks. The framework COSMOS [260] was tested for four scenarios: stable Wi-Fi, indoor Wi-Fi, outdoor Wi-Fi, and outdoor 3G. AD-Hoc Cloudlets oriented computation offloading was tested in [38] in the context of 3G/4G connectivity. Chen et al. [39] have studied the mobility and its impact on the performance of caching and computation offloading in 5G-ultra dense cellular networks. Other studies Considering computation offloading in MEC [321] and Cloud computing [134] in 5G heterogeneous networks are proposed in the state-of-the-art computation

offloading.

3.3.3 Capabilities

Mobile devices have achieved great development in terms of capabilities (i.e., hardware resources) during the past decade.

1. ***Central Processing Unit (CPU)/Graphic Processing Unit (GPU).*** Multi-core processors with a high clock speed are being proposed. For instance, ARM industry developed microprocessor cores with high performances, power and cost requirements, such as *Cortex-A*. Cortex *A7, A9*, and *A15* execute 32-bit instructions while Cortex *A53, A57*, and *A72* process 64-bit instructions[4]. These processors offer better performances, at the expense of energy[5]. *Nvidia's Tegra*, Samsung's Exynos, *Apple's A8*, and *Qualcomm's Snapdragon* are few of chips in the industry. Regarding GPU, mobile devices have been designed with GPU since 2013. *Mali-T604* was the first GPU integrated inside *Samsung Chromebook* and the *Google Nexus 10*[6]. *PowerVR, Tegra*, and *Adreno* are other GPUs developed by Imagination Technologies, Nvidia, and Qualcomm, respectively.
2. ***Battery.*** The energy is the unique non-replenishable resource. It needs an external resource to restore it. A plenty of efforts seeking to harvest energy from renewable resources including human movement [211], solar energy [23, 172], and wireless radiation [113]. In parallel, researchers aim at reducing the energy overhead in different aspects of computing, including hardware, OS, application, and networking interface. DVS technology [28] is a power management technique used to conserve the energy, particularly in wearable devices by increasing and decreasing the voltage. The automatic shutdown of the screen is another concept to save energy for mobile devices. Some other efforts are trying to develop alternative energy resources such as nuclear batteries.
3. ***Screen.*** The screen size has grown since 2007 from $2.59\,inches$ to $6\,inches$ for today's smartphones. For instance, the *iPhone 8 Plus* has a screen size of $5.5\,inches$. Despite, these large screens offer high visualization, they sharply drain the batteries. Some alternatives have been proposed, they include the *GeoTV* [43], and *remote display* solution [264]. Other efforts to extend data-presentation area have considered the utilization of dual screens such as *Codex* [106], *FoldMe* [131], and *Foldable3D* [29].
4. ***Memory and storage.*** Today, most smartphones have a Random Access Memory (RAM) for running applications. Their capacity increased over the year following *Moore's* law from only $100\,MB$ to up to $4\,GB$ today. Regarding the storage, smartphones are equipped with a large flash storage (e.g., $64\,GB$) used to store user's files such as pictures, applications, music, and videos. The speed of evolution of the storage capacity is doubling every two years, fit to Moore's law again.

[4] http://www.arm.com/products/processors/
[5] http://www.arm.com/products/processors/cortex-a/
[6] http://malideveloper.arm.com/

Augmenting the hardware increases the performance of the mobile devices, however, it brings different drawbacks, that include (but are not limited to):

- Increasing the energy consumption due to large screens, powerful processors, and large storage.
- Decreasing the hardiness of mobile devices due to the additional size, weight, and heat, which are caused by the incorporation of powerful processor, large storage, and big screen.
- Increasing the price of these devices due to the integration of high technology and powerful ships in a dense space.

3.4 Computation Offloading Taxonomy: From the Application Perspective

We focus in this section on motivations, issues, and challenges encountered by mobile applications when requesting computation offloading. It will be of special interest to identify the *genre* of applications, described in the literature, which advocates and motivates the use of computation offloading. It is also interesting to identify the application programing languages, aiming at discovering application portability and developing frameworks that target, as much as possible, compatible applications. We are also motivated to classify applications in the literature under different resource demand groups in order to define objective functions that should be considered for each group. We also analyse application code dependency challenges. We represent these different axis in Figure 3.3.

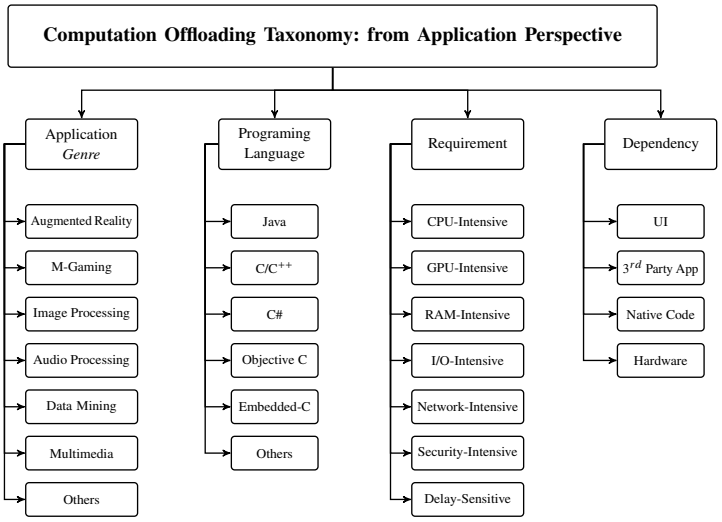


Figure 3.3: **Applications-based computation offloading taxonomy**

3.4.1 Application *Genre*

The study in Section 3.2 has demonstrated that not all mobile applications are suitable for computation offloading, since the offloading gain depends on several metrics (e.g., computation amount, exchanged

data). We report in this section the main suitable applications, described in the literature, for computation offloading.

- **Augmented Reality**. In [249] an AR application is described. It presents eyeglasses for cognitive assistance, composed of a camera and earphones. The system hosts different applications, including AR, computer vision, and audio processing. The authors in [284, 286] present another AR application, which features marker-less tracking and object recognition. The system tracks feature points in a video to enable the overlay of 3D objects. The application is divided into several modules that each can be offloaded to a Cloudlet to optimize the user experience. In [288], a middleware built upon OSGi[7] is presented with an algorithm, which calculates the best deployment of the AR application according to the possible configurations.

- **M-Gaming**. MAUI [59] and ThinkAir [139] have been tested for arcade, chess, and N-Queens puzzle games. MAUI represents the games into a graph, then draws an ILP model to obtain the location of each method. ThinkAir uses past-invocations of methods to to make decisions (see Chapter 5 for more details). Messaoudi et al. have discussed the possibility of offloading game engines in [192], and proposed, in [189], the *UCOF* or Unity 3D-based Computation Offloading Framework. *Cloud gaming* [111] represents another use case of computation offloading. Here, the entire application (i.e., game engine) is offloaded and computed in the Cloud. Commands and video streams are exchanged between the powerless devices and the game engine hosted in the Cloud (more details in Chapter 4).

- **Image Processing**. In [188], a framework has been proposed that relies on the MEC to enhance the execution of a facial recognition application (see Chapter 6). A facial detection scenario was described in [179] wherein the application is represented by an hybrid (i.e., parallel and sequential) dependency graph, which is drawn into an ILP to make offloading decisions regarding energy consumption. Image manipulation based on cyber foraging was proposed in [147]. The use case describes touching an image up by applying filters, sharpening, removing the red-eye effect, and adjusting brightness, colour, and contrast. Other use cases were presented in [125, 127, 130, 310] for optical character recognition (OCR), barcode analysis, face detection, and object recognition.

- **Audio Processing**. Goyal and Carter (GnC) framework was tested for speech recognition applications in [93]. The authors used the *Sphinx2*[8] application. GnC records what the client is saying and sends the sound data to a surrogate, which in turn recognizes the phrases and sends the results back to the client. *Locuts* [146] is a framework providing a tool for developing applications-oriented cyber foraging. A daemon running on the powerless device calculates an optimal execution plan according to the resource measurements on the device and surrogate. The example given describes

[7] `https://www.osgi.org/`

[8] Real-time application incorporating a large American English vocabulary with pronunciation dictionary.

a doctor who enters patients information on his electronic journal using a lightweight headset, which needs a surrogate for translation.

The above-mentioned use-cases/applications are a small sampling of the applications and scenarios used in the literature. The applications used in state-of-the-art computation offloading include, but are not limited to, data mining [93], text translation [310], 3D home interior design [91], virus scan [52, 227], SciMark 2.0 benchmarking [324], multimedia (M4Play) [217], and mathematics processing (π-calculator) [217].

3.4.2 Programing Language

Computation offloading frameworks and mobile applications are highly dependent, as they should support code portability. We distinguish two alternatives: (i) a *single-platform*, which is a light-weight, efficient approach, wherein the offloading framework is dedicated for applications developed on the same environment; (ii) a *Cross-platform*, the framework of which can run on several operating systems and various hardware configurations. Therefore, a cross-platform framework supports a variety of applications written in diverse programming languages. However, this solution is complex, and it induces a high overhead due to wrapping the application components in order to be supported. To sum up, application that are candidates for offloading depend on their frameworks, which should be portable through platforms.

Table 3.2 summarizes some reviewed works. We observe that most of the proof-of-concept computation offloadings are based on Java applications that encompass facial detection and recognition, speech translation and recognition, and 2D games. Researchers tend to focus on Java as it is easy to use (i.e., it is a high-level language), is supported by the main mobile operating systems (such as Android), includes memory management and a garbage collector to efficiently manage the memory, and it defines a set of services and APIs that are ready for use. In contrast, C/C++ and embedded C are low-level programing languages that involve considerable efforts from developers to manage the memory and develop applications.

3.4.3 Requirements

Below, we classify the requirements of candidate applications for computation offloading into seven groups, according to the needs served by the applications. This classification is useful for managing the framework objective functions through the applications requirements.

- *CPU-intensive* category. This category includes applications that need powerful CPUs to execute and consume a consider-able amount of energy. We mention audio analysis, image processing, data analysis, and benchmarking.

Table 3.2: **Applications with programming language used in computation offloading arena**

Framework	Description	Application	Language
Siri	Apple's commercial framework which relies on the Cloud to get things done	Speech recogn	Objective C
MAUI [59]	Leveraging the Cloud to enhance dynamic offloading of methods	. Games . Face recogn	.Net
VM-based Cloudlet [99]	Uses VM-based Cloudlets with a delivery approach for hostile environment	OpenCV	C++
CloneCloud [50]	Dynamic offloading of threads from mobile devices to VM clone in Cloud	. Virus scan . Image search	Java
ThinkAir [139]	Offloads methods at runtime to a VM clone in the Cloud	N-Queens puzzle	Java
Scavenger [147]	Offloads tasks to surrogates based on dual scheduling profiles	Image decorator	Python
Spectra [82]	Proposes a framework to develop application-oriented offloading	. Speech Recogn . File Prepar . langauge Transl	. NA . Latex . NA
Cuckoo [128]	Partial offloading with programming UI to ease the code integration	. Object recogn . Face recogn	Java
eXCloud [176]	Uses stack-on-demand technique to offload segments of frames to the Cloud	Math calculations	Objective C
Zhang et al. [322]	Weblets-based partitioning using elastic applications approach	Image processing	C#
Giurgiu et al. [91]	Distributes application into functional layers based on the R-OSGI	3D home interior	OSGI (Java)
Ou et al. [217]	Adaptive $(k+1)$ graph partitioning algorithm for offloading	MPEG-4 player	Java
Verbelen et al. [283, 286]	Leverages Cloudlets for immersive applications	AR	C/C++ wrapped in OSGI
DPartner [323, 324]	On-demand automatic factorization for android bytecode	. Linpack Bench . Gomoku Chess . 3D car Games	Java (bytecode)
JDOP [296]	Dynamic graph partitioning algorithm for Java bytecode applications	. Java Grand . SpecJVM98	Java (bytecode)
Chroma [18]	Proposes a tactics-based remote execution for mobile computing	. Pangloss-Lite . Janus . Face	. C++ . C . Ada
Yang et al. [310]	Leverages surrogates for bytecode offlaoding	AutoTranslator	Java (bytecode)
Coca [37]	Offloads computation to the Cloud using aspect-oriented programming	Chess game	Java
IC-Cloud [261]	Predicts connectivity on intermittent Cloud to enhance offloading	. Face Detect . Voice recogn . Droid Fish	Java

- *GPU-intensive* category. The GPU, as its name implies, is largely dedicated to graphic processing, so applications with intense rendering are included in this category. Games, AR, VR, and multimedia are the main applications.
- *RAM-intensive* category. Some applications demand much memory to run, such as 3D modelling and gaming. Also, the IoT, including sensors, are memory-constrained devices. Therefore, almost all programs running on these devices are RAM-intensive.
- *I/O-intensive* category. This category contains the applications that increase memory paging and I/O activity such as PostMark[9], PageBench[10], and Bonnie[11].
- *Network-intensive* category. This category includes applications involving Internet connectivity used to access remote services. We mention social network applications, such as Facebook, and

[9]File system benchmark program: `http://www.netapp.com/tech_library/3022.html`.
[10]Synthetic program which initiates and updates an array whose size is bigger
[11]Unix file system performance benchmark: `http://www.textuality.com/bonnie/`.

Twitter, *Sftp*[12], and *PostMark_NFS*[13].

- *Security-intensive* category. This category concerns applications that need security mechanisms, such encryption, authentication, and authorization.
- *Delay-sensitive* category. These are mainly considered to be real-time applications, such as games and AR/VR.

In general, memory [319], CPU, and GPU high-demand applications are mainly suitable for computation offloading. It is prudent to execute bandwidth-intensive applications locally at the mobile device [116]. Offloading delay-sensitive applications [204] depends on the network latency and data exchange: (a) the higher the network latency, the lower the offloading gain; and (b) the larger the exchanged data size, the higher the latency, and the lower the offloading gain. In mobile networks, the latency is impacted by different parameters, which depend on both the core network and wireless channel access delays. Kumar et al. [150] have shown, via a mathematical model, that if the access delay is higher than the execution time on the mobile device, then, even if the server is infinitely fast, there is no offloading gain. Concerning the network-intensive [20], the I/O-intensive [319], and the security-intensive [33, 132, 270] applications, their execution in remote servers is restricted by some constraints including, hardware dependency, security and confidentiality, and network communications. Almost all the applications reported in the manuscript are computationally-intensive applications.

3.4.4 Code Dependency

We finish this section by distinguishing components of applications that cannot be offloaded. We call this constraint *code dependency*. For some reasons, which we will discuss later, offloading the entire application is not always possible, as some components of the mobile application depend on specific hardware components, such as sensors or on the device software, such as libraries. Figure 3.4 portrays some of the different hardware and software entities involved in an interaction between a user and his device. Obviously, in reality, the device interacts with the user via multiples of these entities. As this figure shows, a user application, from the background, *may* depend on diverse programs, known as *libraries*, *Software Development Kits (SDKs)*, *middleware*, *operating systems*, and *drivers*, but also on hardware, especially sensors. From the foreground, the application interacts with the user via *User-Interfaces (UIs)*, which manage for instance, the mouse and touch screen for the inputs, and headphones and speakers for the outputs.

We now clarify, the role of the layered programs triggered in an application execution.

- *The platform independence layer* is a set of libraries that allow an application to be *cross-platform*. It sits atop the hardware, drivers, operating systems, and third-party SDKs. It releases the rest of the application from being dependent on the underlying platform.

[12] Synthetic program which uses sftp to transfer a 2GB size file
[13] The Postmark benchmark with a NFS mounted working directory

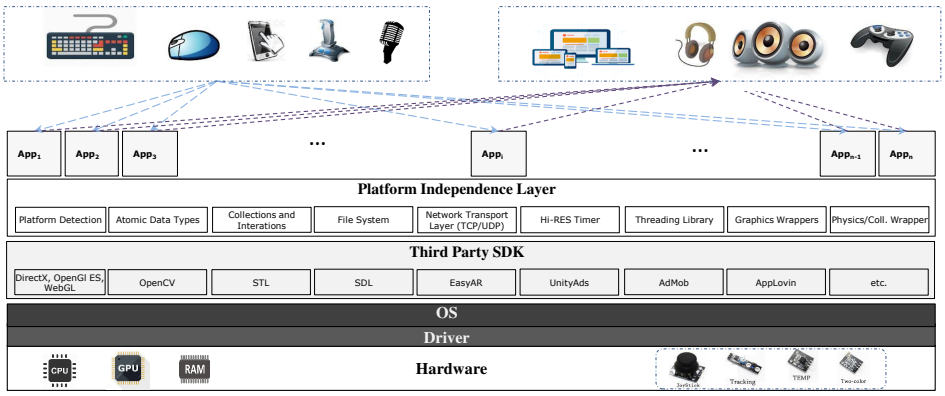


Figure 3.4: **Hardware/software entities involved for human-machine interaction**

- *The third-party SDKs and middleware* are APIs leveraged for a large number of applications. To name a few examples: *OpenGL ES*, *PhysX*, and *STL*.
- *The operating system* is the software that runs continuously on a device to orchestrate the execution of multiple applications, one of which is the application requesting computation offloading. The software uses interruptions and preemption scheduling to share the hardware with the applications that are running.
- *The device drivers* are low-level software components provided by the operating system or the hardware vendor. They manage hardware resources and release the operating system and upper layers from the details in communications with the hardware.

Accordingly, we can classify the non-transferable components of an application into four groups: (i) components involving UIs [59, 216, 217]; (ii) components interacting directly with the device hardware, such as a Global Positioning System (GPS) and accelerometer sensors [216]; (iii) components depending on APIs, middleware, or libraries located on the same device [94, 220]; and (iv) components using directly device-related information [94].

3.5 Computation Offloading Taxonomy: From the Network Perspective

This section provides some background regarding the network environment in which, the powerless device and server are performing the computation offloading. Particularly, in this section, we focus on the QoS metrics, communications support, and networking models. Figure 3.5 depicts these points.

3.5.1 Metrics

- *Delay*: The main aspect of seamless application execution is the interactive delay. Interactive delay is defined as the elapsed time between the moment when an action is triggered by a user and the moment when the result of this action is perceived by the user. Interactive delay is a mandatory

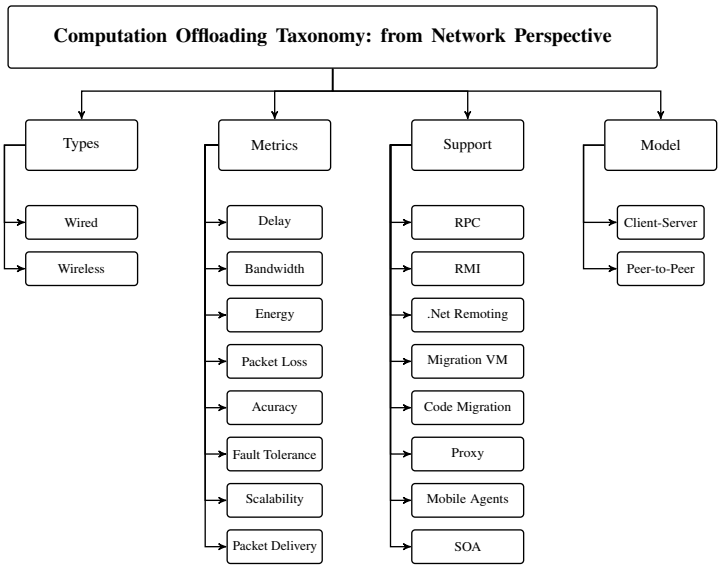


Figure 3.5: **Network-based computation offloading taxonomy**

constraint in MCC, since it is linked to interactivity. The work in [279] demonstrated that a user is satisfied and productivity is not affected by an interactive delay when the latter is below $150\,ms$. The user becomes aware of an interaction delay within the range of $150\,ms$ and, furthermore, the user becomes unsatisfied and frustrated for delays higher than $1\,s$. Work in [45] decomposed the interactive delay into six parts: $t_{client} + t_{access} + t_{isp} + t_{transit} + t_{datacenter} + t_{server}$, wherein t_{client} is the *playout delay*, which is the time spent triggering an action, and receiving and visualizing the result; t_{server} is the *processing delay*, which is the time spent by the server in computing the incoming tasks from the client and transmitting back the results of the tasks; and $t_{access} + t_{isp} + t_{transit} + t_{datacenter}$ represents the *network latency*, where, t_{access} is the data transmission time between the client and the first router, t_{isp} is the phase between the access router and the access point, $t_{transit}$ is the delay in reaching the front-end of a datacenter, and, $t_{datacenter}$ is the time to access the server reserved for the client. To reduce the interactive delay, several works related to computation offloading have focused on bringing the Cloud closer to the mobile user by leveraging the MEC [169, 247, 315], Fog computing [40, 101, 314], and Cloudlets [250, 271, 272] paradigm.

- *Bandwidth*: In a packet network, both bandwidth and throughput characterize the amount of transferred data over the network per unit of time. Bandwidth estimation is of interest to users aiming to optimize the end-to-end transport performance, overlay network routing, and peer-to-peer file distribution. For several data-intensive application, such as multimedia streaming and file transfers, the available bandwidth impacts directly the performance of the application. Three main metrics characterize the bandwidth, namely, *capacity*, *available bandwidth*, and *Bulk Transfer Capacity (BTC)* [228]. Several bandwidth estimation methodologies have been proposed in the

literature to estimate the end-to-end capacity and available bandwidth. We cite the *variable packet size probing* [68, 152], the *packet pair/train dispersion probing* [25], the *self-loading periodic streams* [117], and the *Trains of Packet Pairs* [186] methodologies. Various strategies have been used to improve the computation offloading performance by using high bandwidth link [78,326], on-demand bandwidth allocation [116], multi-Radio Access Technology (RAT) [178], and Multipath TCP (MPTCP) [177].

Several other metrics are considered in computation offloading, we cite energy consumed by each communication technology (3/4/5G, and Wi-Fi), packet loss, accuracy, fault tolerance, scalability, and packet delivery [161, 196, 272, 287].

3.5.2 Communication Support

To perform a computation offloading, communications between a mobile device requesting offloading computation and a remote infrastructure, which computes the offloaded code, have to be established. We distinguish several mechanisms by which to communicate with state-of-the-art computation offloading, such as; RPC, RMI, VM migration, Mobile agents, and proxy.

- *RPC*: RPC is a request–response protocol. The request is initiated by a mobile device, which sends a message to a well-known remote server requesting the execution of a specified procedure with supplied parameters. The answer to the request is performed by the remote server. During the remote execution, the mobile device is in waiting mode until the reception of the response from the server. In this mechanism, applications are partitioned into locally executable code and remotely executable services. These services are pre-installed on remote infrastructure, which expose these RPC as APIs. Several frameworks have been proposed in the literature for computation offloading based on RPC support for communication. We mention *Spectra* [82], *Chroma* [17,18], *MAUI* [59], and *Odessa* [231].
- *RMI*: RMI is a Java API that performs a remote invocation of methods; it is the object-oriented equivalent of RPC. RMI supports a direct transfer for serialized Java classes and distributed garbage collection. RMI invokes remote methods in a transparent and seamless way using references for the remote objects. To perform RMI, two particular classes should be created by the Java Development Kit (JDK): the *stub* from the client side and the *skeleton* from the server side. These two classes are in charge of calls, communications, execution, and sending and receiving execution results. Past studies such as *Cuckoo* [128], *JavaParty* [226], *J-Orchestra* [278], and *JADE* [11] have been designed with the RMI mechanism.
- *VM Migration*: VM migration is another mechanism that can be used to support communications for computation offloading. The concept is to create a VM that replicates the execution environment of the mobile platform, and migrate the VM to a remote server to compute the running application. VM migration follows two ways; *cold*, and *hot* migration. The difference between these two

approaches is in how data are migrated. In cold migration, the VM is stopped to migrate all the pages related to the VM, while in the hot migration, the VM is migrated without interrupting the OS or any of its applications, giving the illusion of seamless migration. Some research on state-of-the-art computation offloading have been conducted using virtualization and VM migration. We cite *CloneCloud* [50, 51], *GnC* [93], *Slingshot* [275], and *Cloudlet* [249].

- *Mobile Agents*: Mobile agents are autonomous programs which are able to control their movement between machines in an heterogeneous network. Communication between agents is achieved through message passing using RMI or RPC. In [11], the authors used the Java Agent Development Environment (JADE)[14]. JADE is a popular framework for developing agent applications for interoperable, intelligent, multi-agent systems. Scavenger [147] is another framework, in which surrogates run a daemon, which is responsible for offering remote access for the mobile code execution environment and device discovery.

3.5.3 Network Type

- *Peer-to-peer*: In a P2P topology, the overall system load is distributed among all participating mobile devices/peers, which organize themselves to act as clients and servers at the same time. In the computation offloading umbrella, several works have been proposed in the peer-to-peer context. In [140], the authors have introduced the *Clone2Clone (C2C)*, a distributed peer-to-peer platform for Cloud clones of smartphones. Chen et al. [38] have proposed a peer-to-peer model to interconnect nearby mobile devices through various short-range radio communication technologies forming ad-hoc Cloudlets, where every mobile device works as either a consumer or provider of a service. In [110] a peer-to-peer model is described for Cloud robotics.

- *Client-Server*: In client-server model oriented for computation offloading, the client represents the powerless device such as smartphone, while the server generally, corresponds to a resourceful machine able to compute the client's requests, for example, server in the Cloud or Grid computing. In [239] a stochastic model of the client-server system based on Markovian decision processes wherein the power management problem is formulated with task migration as an optimization problem. An energy trade-offs in offloading computation/compilation in Java-enabled mobile devices [36] have been studied considering the client-server model. To summarize, most of the research works presented through this document are based on the client-server model.

3.6 Computation Offloading Taxonomy: From the Server Perspective

In this section, the focus is on servers type, their distance by report from mobile users, their capabilities, and their associated issues. We review state-of-the-art regarding these axes and compare some frame-

[14] `http://jade.tilab.com`

works used in the literature for computation offloading. The aim is to see where we can contribute and how some challenges are resolved. Figure 3.6 summarizes this section.

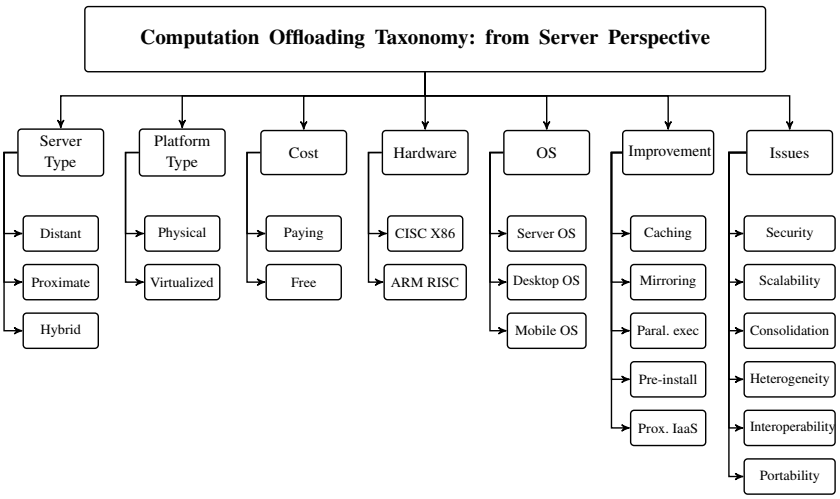


Figure 3.6: **Server-based computation offloading taxonomy**

3.6.1 Server Type

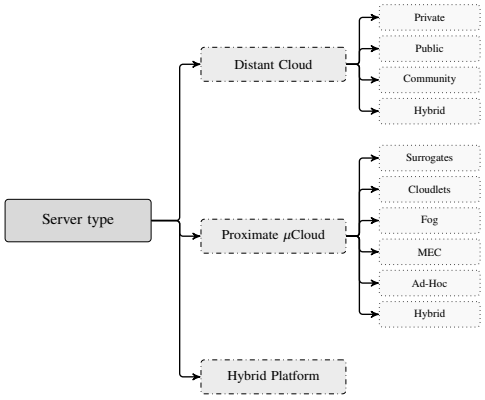


Figure 3.7: **Taxonomy around server type**

This section reviews the platforms used to compute offloaded tasks. We attempt to classify these platforms according to their physical distance from mobile users and their capabilities, into three categories, namely, *distant Cloud*, *proximate μCloud*, and *hybrid platform*, as shown in Figure 3.7 and described below.

1. Distant Cloud.

   The Cloud is a model that enables ubiquitous, on-demand access to a shared pool of configurable resources (including networks, storage, applications, and services). The Cloud offers three types of services: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)*, and *Infrastructure as*

a Service (IaaS). SaaS provides a plethora of remotely accessible applications running on the Cloud infrastructure. PaaS allows customers to deploy applications, libraries, services, and tools supported by the provider. IaaS provisions the consumer with a pool of computing resources. We distinguish four possible deployments of The Cloud as: private[15], public[16], community[17], or hybrid Cloud[18]. In these deployments, public and private stationary servers are delivered by farms and enterprise premises. Amazon[19], Google[20], and Windows Azure[21] are the pioneer providers of Cloud computing infrastructure. Usually, the Cloud is provided in a *pay-as-you-use* fashion.

2. Proximate μCloud.
   In state-of-the-art computation offloading, we distinguished five appellations of platforms located in the near vicinity of mobile users. These platforms cross each other and *may* share some equipment.

   a) *Surrogates* are static, idle computers located in proximity to the mobile users, such as those in airports, cinema halls, shopping malls, and public computing kiosks [85]. The surrogates are extended via high computing capabilities to show advertisements, play music, and compute services and applications. They are plugged to a source of energy and connected to the internet.

   b) *Cloudlet* was proposed by Satyanarayan in [250], and it extends the definition of surrogates. It is a resource-rich computer or a cluster of computers, seen as a “data-center in a box" with self-managing that requires internet connectivity, power, access control, and some security mechanisms. It is connected to mobile users via a one-hop, high-speed LAN. Cloudlets are located in public places (e.g., coffee shops or libraries), at access points (e.g., boxes and gateways), or in homes (e.g., desktops).

   c) *Fog* is an extension of Cloud computing, which brings network resources from the core network to the edge network to reduce the latency. It is gaining a lot of industry support, particularly from Cisco[22]. For instance, Fog could be the Mobile Network Operators (MNOs) infrastructure scattered in urban area [246]. Fog is a highly virtualized platform, which provides a wide range of applications and services in highly distributed deployments and a real-time manner, such as gaming, video streaming, and AR/VR.

   d) *MEC* mainly targets 3GPP-based mobile networks. It provides computing capabilities within the RAN, in close vicinity to mobile users. It offers ultra-low latency with high-bandwidth. Various use cases are leveraging MEC, such as location tracking, AR, video analytics, and

[15] The IaaS is provisioned for single organization.
[16] The IaaS is provisioned for an open use.
[17] The IaaS is provisioned for a specific organization to ensure concerns (e.g., mission, security requirements).
[18] The IaaS is a composition of distinct Cloud infrastructures.
[19] `https://aws.amazon.com/`
[20] `https://cloud.google.com/`
[21] `https://azure.microsoft.com/en-us/`
[22] `https://developer.cisco.com/site/iox`

distributed content. MEC is heavily promoted by the ETSI, which is trying to develop a standard around MEC [73], defining a reference architecture and a set of APIs[23].

e) *Ad-Hoc platform* refers to clusters of mobile devices, including smartphones, tablets, and laptops. These devices are collaborating in an heterogeneous platform [245] to compute tasks for a nearby mobile user. The Ad-Hoc platform provides a short latency and an heterogeneous computing environment. However, the ad-hoc capabilities are generally limited, and hence, are unable to perform intensive computational tasks [180]. Moreover, scheduling, mobility, security, and privacy present other issues.

3. Hybrid Platform.

The hybrid infrastructure is a mixing of proximate devices, be they mobile or immobile, with the distant Cloud as depicted in Figure 3.8. The goal is to strike a balance between network latency, server capabilities, and server availability [235]. The main idea is to offload the delay-sensitive tasks to the proximate infrastructure, and the CPU-intensive tasks to the distant Cloud. Such infrastructures are useful in terms of maximizing the benefits of offloading computation. They deliver enhanced security and privacy features and increase the QoS. Mobile users can reap advantages from these features. However, due to its mobile nature, the deployment, management, and resource scheduling of such an infrastructure is not a trivial task.

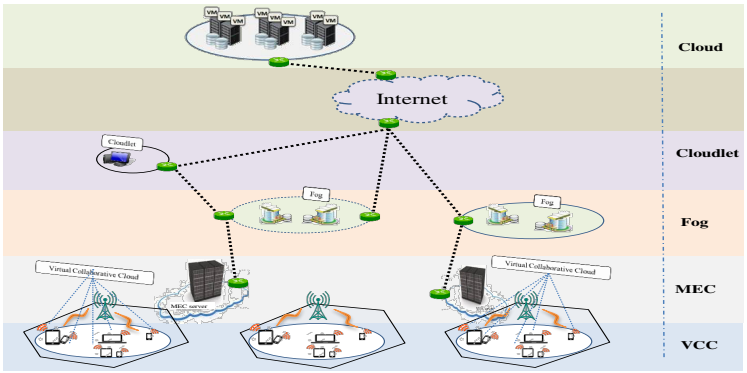


Figure 3.8: **Hybrid Cloud model**

Comparison.

Table 3.3 compares the three aforementioned platforms based on several characteristics, while Table 3.4 reviews various frameworks from the server perspective. In the table, symbols: ●, ■, and ▲ mean, respectively, low, medium, and high.

[23] http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing

Table 3.3: **Server types characteristics' comparison**

Characteristic	Cloud	Surrogates	Cloudlets	Fog	MEC	Ad-Hoc	Hybrid
Communication	3G/4G/5G/WiFi	3G/4G/5G/WiFi	3G/4G/5G/WiFi	3G/4G/5G/WiFi	3G/4G/5G/WiFi	3G/4G/5G/WiFi	3G/4G/5G/WiFi
Bandwidth	Moderate	High	Moderate	High	High	High	Moderate
Latency	High	Low	Low	Low	Low	Low	Moderate
Architecture	Distributed	Distributed	Distributed	Distributed	Distributed	Distributed	Distributed
Capabilities	High	Medium	Medium	Medium	Medium	Low	High
Flexibility	High	Medium	Low	Medium	Medium	High	High
Availability	High	Medium	Low	Medium	Medium	Medium	High
Scalability	High	Medium	Low	High	Medium	Medium	High
Cost	Pay-As-You-Use	Pay-As-You-Use	Pay-As-You-Use	Pay-As-You-Use	Pay-As-You-Use	Credit / Reward	Infra. Dependent
Heterogeneity	High	Medium	High	High	Low	High	High
Security	High	Moderate	Low	Moderate	Moderate	Low	High

3.6.2 Server Improvements

In this section, we present several improvements that, when applied to the remote servers, increase the offloading gain by reducing the network latency and execution time.

1. **Caching**. Caching refers to the storage of data near the mobile user in order for it to be used in the future. We distinguish four categories of caching: (i) *software package caching* [93], in which software packages are cached for future installations to avoid download delays; (ii) *service code caching* [143], which is used to store code sources in jar files; (iii) *VM synthesis caching* [249], which is employed as a pre-fetching technique to reduce VM synthesis delay; and (iv) *data caching* [79], which stores data, such as video streams, near the mobile user in what are known as Content Delivery Networks (CDNs). Caching decreases network latency and reduces redundant traffic. However, it increases the overheads, cost, and consistency issues.

2. **Parallel execution**. Parallelism is the simultaneous execution of different program sections. It is designed in two separate and complementary functionality: computation, which expresses calculations in a procedural manner, and synchronization, which abstracts communications and concurrency controls. Parallelism in the Cloud is viewed from two perspectives: (i) *intra-server parallelism*, wherein the server hosts multiple VMs. In addition to performance improvement, this approach reduces server authentication and selection time; (ii) *inter-server parallelism*, which performs parallel execution on multiple servers, each of which hosts one VM. Each server mandates an authentication to deploy a VM, which induces high network overhead and operational complexity. Kosta et al. [139] devised a framework that uses these two approaches, and Zhang et al. [322] proposed to design elastic mobile applications using weblets distributed over multiple servers. Despite the performance improvement and the scalability amelioration, parallelism increases power consumption, mandates authentication, and requires more hardware.

Table 3.4: **Framework comparison according to server type**

Framework	Description	Server	CPU Augmentation	Battery Prolonging	RAM Extension	Overhead	Security	Responsiveness	Complexity
VEE [116]	The system clones Android mobile platforms into VMs in the Cloud. Both the VM and the mobile device are synchronized to keep both copies of the application updated by sending only the segment of the data stack that is created by the application. The system stores input events using a record/reply mechanism with pseudo checkpoint methods, the purpose is to increase the quality and efficiency of computation offloading. Doing such, involves efforts from the application developers to specify the global and local states of the application.	Cloud	■	■	NA	■	●	■	■
CMH [181]	Application in CMH model are developed in two steps; the heavy components are developed for the Cloud-side execution, while the lightweight and native code of the application are developed for mobile device execution. The CMH framework does not need for profiling, partitioning and code migration, hence the system induces least overhead. Once the computation on the Cloud is finished, the results are send back to the mobile device for integration with the other components. The application developing is complex due to the interoperability, heterogeneity, and vendor's lock-in problems of infrastructures.	Cloud	▲	▲	▲	●	NA	■	▲
Kun et al. [310]	The authors have proposed a framework for offloading intensive tasks. The offloading process follows the traditional steps of offloading (i.e., resource discovery, profiling and monitoring, modelling, partitioning, and communication). The framework operates at the Java class granularity level. The proposed model is a multi-cost graph. Partitioning the graph, is a $(k+1)$ graph partitioning algorithm, which finds k $(k \geq 1)$ remote partitions and one local partition.	Surrogates	▲	▲	NA	▲	NA	▲	■
Scavenger [147	*Scavenger* [147] is a cyber foraging system written in Python . It is composed of two independent software components: a daemon running on surrogates, and the library used by the client applications. The objective of Scavenger is to improve the performance by dynamically offloading tasks to surrogates. On the surrogates, tasks are computed in a *round robin* fashion.	Surrogates	▲	▲	NA	▲	NA	▲	▲
Cloudlet [250]	Delivering a VM inside a Cloudlet have been proposed. The framework is based on the *dynamic VM synthesis* approach.	Cloudlet	■	■	■	■	■	▲	●
Verbelen et al. [284]	Authors propose to alleviate the issue of Cloudlet deployment dependency on service providers, using a dynamic Cloudlet concept. Also, they solve the performance and flexibility issues of VM-based Cloudlets by proposing a dynamic offloading at component granularity. The framework distribute the application components among nodes in two Cloudlets. All components are managed and monitored by an execution environment, which in turn is managed by a node agent that communicate with Cloudlet agents. The authors have used the OSGi framework	Cloudlet	▲	▲	NA	●	NA	▲	■
Smart GW [4]	Authors propose a smart gateway, accompanied with Fog computing. The smart gateway is presented in layered architecture, each layer performs a job. The gateway includes a virtualized layer that hosts virtual (network) sensors on the physical layer. The functions of the gateway are shared among activities and resource monitoring, data analyzing, filtering, and trimming, temporary storage, security concerns, processing IoT tasks or offloading them the the Fog.	Fog	▲	NA	NA	●	▲	■	■
Mohamed [102	To make a proper offloading decision, authors have executed the same application task under three virtual Fog, and two virtual Cloud infrastructures, then identified the parameters impacting the offloading performance. The problem was formulated, using a constrained graph partitioning problem. The authors used the method granularity level for offloading.	Fog	▲	▲	NA	■	▲	▲	■
Liu et al. [169]	A power-constrained delay minimization problem for computation offloading based on Markov decision processes has been proposed and adapted to the MEC context. The authors have considered a mobile device running computation-intensive and delay-sensitive applications. The framework considers the mobile device as a composition of a task buffer, a transmission unit, and a processing unit. Authors propose an algorithm based on the average delay and power consumption at the mobile device for each task to solve the optimization problem.	MEC	▲	▲	NA	▲	NA	▲	▲
You et al. [315]	A multi-user resource allocation for MEC has been also proposed. The model is formulated as a convex optimization problem to minimize the mobile energy consumption considering the computation overhead and the capacity of the MEC. The model derives an offloading priority for each user according to its channel gain and energy consumption. A low priority derives a minimum offloading, while a high priority performs a complete offloading.	MEC	NA	▲	NA	▲	NA	NA	▲
MOMCC [8]	The framework uses the SOA, and leverages a cluster of nearby mobile devices to compute the offloaded tasks in a collaborative way. An application is a set of services, developed independently from others. Several smartphones, in near vicinity, are collaborating to compute the services, and therefore earn some money. To offer an IaaS, mobile devices register with UDDI and negotiate the services to compute. Several issues should be considered including, resource limitation, security, and mobility of users.	Ad-Hoc	▲	▲	▲	●	■	▲	■
VMCC [114]	Leverages an ad-hoc cluster of nearby smartphones as an infrastructure. The application code is updated with a proxy and RPC instructions. For each application, the framework needs to determine the number of smartphones needed, the offloading overhead, and the security requirements. Hence, the system incessantly traces the smartphones and their geolocation. Upon the partitioning step, the remote partition composed of small codes is offloaded among the available smartphones. The results are integrated back in the mobile device upon completion.	Ad-Hoc	■	■	■	■	■	▲	■
SAMI [246]	The infrastructure is composed of Cloud, nearby MNO, and cluster of very close MNOs authorized dealers. This architecture proposes a multi-tier IaaS that aims to improve the performance, increase the offloading flexibility, and save energy on the mobile devices. MNO dealers compute the latency-sensitive tasks. Depending on the resource scarcity and user mobility, tasks can be computed on the MNOs or on the Cloud. The resource allocation is done by an arbitrator entity, which considers several metrics including resource requirements, latency, and security requirement of services.	Hybrid	▲	▲	▲	●	■	■	▲
MOCHA [272]	Combines Cloudlets and Cloud to perform a computation offloading. The mobile device sends to the Cloudlet the heavy tasks, the latter partitions these tasks between itself and the Cloud to improve the QoS. MOCHA uses two partitioning algorithms; *fair distribution* of computation among Cloudlets and Cloud, and *greedy distribution*, wherein tasks are distributed between the Cloudlets and the Cloud according to their responsiveness.	Hybrid	■	■	■	NA	NA	■	■

3. **Mirroring**. A mirror server is a configured server that hosts multiple VM templates with default settings to serve a large number of mobile users. Mirroring offers three services: file scanning, file caching, and computation offloading. Computation offloading induces a high overhead due to the VM cloning, migrating, and configuring on the mirror server. Zhao et al. [326] devised and tested a framework, *Mirror Server*, employing Telecommunication Service Provider (TSP)-based remote services. The framework provides a lightweight protocol for accessing remote services on the mirror server and uses an optimized mechanism to download and offload applications. Mirroring offers high scalability, but limited services and low security, as TSP mirror servers are not designed for data processing in the way that Cloud is.

4. **Pre-installation**. Installing software packages or complete application on remote servers before running the application improves the application performance. Since the application and packages are already installed on remote servers, initiation, preparation, and migration delays are avoided. We distinguish two approaches; the first relies on the pre-installation of root partitioned images with applications, operating system boot-up scripts, and software packages [93]; while the second approach provides a pre-installed run-time environment to ease the deployment of mobile applications in the Cloud [154]. Pre-installation decreases network communications and delay as there is no need for application migration. However, it wastes resources when applications are not used and requires maintenance.

5. **Leveraging proximate infrastructure**. Proximate infrastructures includes Cloudlets, MEC, and Ad-Hoc Cloudlets. The aim is to reduce the WAN latency when leveraging Cloud resources. Chapter 6 reviews some research works leveraging proximate infrastructures, particularly MEC. In addition to the low network latency induced by these platforms, leveraging proximate resources minimizes the data transport cost. However, proximate infrastructures require isolation to prevent users from unauthorized access. Moreover, deployment requires investigations into illegal and opportunistic access to these proximate platforms and negotiating an SLA between the mobile user and the infrastructure provider.

3.6.3 Platform Type

The IaaS oriented for computation offloading is presented to mobile users in the following two forms.

- *Physical resources*: These resources concern powerless mobile devices and some Cloudlets, which do not support virtualization technology [203].
- *Virtual resources*: Mobile users offload computation to servers in the Cloud, Fog, or MEC. These servers can perform several tasks in parallel. Therefore, a consolidation is mandatory for such resources. Usually, infrastructure providers, such as Amazon, provide resources in the form of VMs.

There are several implications to virtualization-based approaches in computation offloading [155, 263]. Indeed, virtualization is only feasible in high-bandwidth networks. It induces an overhead compared to native executions of applications [263]. A mobile device should support virtualization technology, which is not the case for smartphones and tablets. Virtual Machine Managers (VMMs) should be able to suspend, resume, and migrate VMs; on the other hand, the infrastructure should have enough resources to successfully launch the VMs and execute applications [316]. Several virtualization techniques have been used for computation offloading, which include:

1. *VM Migration*: Some VMMs support VM migration, such as OpenStack[24], KVM[25], and Xen[26]. In VM migration, applications are encapsulated inside VMs and migrated (i.e., pages are transferred) to a remote infrastructure [120]. After processing the applications, the VMs are migrated back to mobile devices (particularly laptops). Techniques, such as *compression* and *write throttling* [262], have been used to minimize the impact of the VM size on the bandwidth-scarce-mobile devices. The ThinkAir framework [139] uses VM migration for computation offloading.
2. *VM Overlay*: This refers to *Dynamic VM synthesis*, proposed in [250] for Cloudlets. The Cloudlet creates VMs dynamically. A small VM overlay, such as a bootstrap configuration file, is delivered to the Cloudlet, which possesses the base VM from which the overlay is delivered. The overlay is applied to the VM, which starts the execution form an identical state to that defined in the overlay. This approach requires a base VM, a VM customization script (VM overlay), and a VM clean-up script.
3. *On-Demand VM provisioning*: The VM is created on demand to satisfy the requirements of mobile devices [70]. The mobile device transfers a provisioning script to the discovered infrastructure, such as a Cloudlet. Then the latter composes a VM that should fulfil the requirements of the mobile device. The Cloudlet has two choices: (i) select an already configured VM from the VM repository that hosts the necessary components, or (ii) select a base VM and install the required components [295]. On-demand VM provisioning differs from VM overlay as, in the latter, the infrastructure hosts only the base VM, and the overlay is provided by the mobile device; while in on-demand VM provisioning, the infrastructure hosts the base VM, server components, and provisioning software.

3.6.4 Issues

Several research works have contributed to infrastructures being efficiently leveraged for computation offloading. However, various issues and challenges have been raised. These challenges, addressed in the literature, are still open. In the following, we highlight most of them:

[24] `https://www.redhat.com/en/topics/openstack`
[25] `https://www.linux-kvm.org/page/Main_Page`
[26] `https://www.xenproject.org/`

- *Security*: Security is central in Cloud-oriented computation offloading. Cloud security issues have been discussed in [132, 305], they include: confidentiality, integrity, availability, accountability, and privacy of data. In MCC, security should be considered from both the mobile device and Cloud sides. In terms of the mobile users, mechanisms should be defined to detect malicious codes (e.g., viruses, and Trojan horses). These mechanisms should run incessantly, which would drain considerably the mobile device's battery. Some research works [210, 298] have been introduced to perform computation offloading of malicious code scanning for security and energy-saving objectives. When it comes to the Cloud, stored data can be lost, altered, or denied. Several proposals for security of data have been put forward, which include *homomorphic encryption* [87, 88], *reputation-based trust establishment* [250], and *authentication mechanisms* [93]. Other proposals have been made in [44, 297, 298], which aim to protect outsourced date. Despite security and privacy issues, these mechanisms induce delay due to authentication and cypher execution.
- *Scalability*: Scalability refers to unbounded resources purchased in several quantities at any time. It ensures service provisioning regardless of the number of mobile devices, VMs allocating the necessary computing resources, and VM migration services being performed for load balancing [14, 238]. Some frameworks, such as CloneCloud [51], Cloudlet [249], Kumar and Lu [151], and Misco [67], are deficient frameworks in terms of a centralized management of the distributed platform. The unavailability of centralized resources is a challenging research issue for ad-hoc and Cloudlets-based systems [114, 170]. Some other frameworks have addressed the scalability issue. *DISC* [9] have been proposed to achieve elasticity for data-intensive service computing. Rai et al. [236] proposed a resource allocation scheme by mapping Cloud resources to a single multidimensional resource vector. *Zeta* [104] is a scheduling algorithm that improves the response quality to meet QoS with few resources. *CloudScale* [258] is another framework designed to manage resource elasticity and scalability.
- *Consolidation*: Resource consolidation consists of using virtualization technology to instantiate multiple VMs. The VMs are dynamically mapped onto a pool of physical resources and concurrently sharing the server resources through the hypervisor. Consolidation improves the utilization of server resources and reduces energy consumption [105]. Consolidation depends highly on the resource variability of mobile applications. Maximizing resource utilization between a maximum number of customers without breaking the users' SLA is challenging. The Cloud provider should define a trade-off between high consolidation and high performance. He should face with the resource demand variability for a single customer and between multiple customers to handle resource peak utilization. In other words, the Cloud provider should design a system to make server consolidation more efficient and less risky.
- *Heterogeneity*: Numerous infrastructure providers expose various services (IaaS, PaaS, and SaaS), making infrastructure heterogeneous. Interoperability and portability are two prominent challenges caused by the heterogeneity of Cloud infrastructures [107]. Cloud business competition is the main

cause of this diversity [69]. A variety of hardware with different architectures between ad-hoc systems, Cloudlets, MEC, Fog, and the Cloud causes the heterogeneity issue for computation offloading, especially when relying on hybrid infrastructures. Connecting heterogeneous systems (i.e., interoperability) is challenging due to the absence of interfaces and standards between infrastructures. In addition, code portability is not an easy task.

3.7 Computation Offloading Taxonomy: From the Framework Perspective

In what follows, we describe state-of-the-art computation offloading from frameworks point of view. The taxonomy proposes seven axis. The first one (Section 3.7.1) is related to code annotation, in which we describe how to annotate the programs of applications to distinguish the anchored from movable code. In Section 3.7.2, we locate the offloading framework either on the mobile device or on the server. Section 3.7.3 distinguishes whether data are present on the mobile device, on the server or saved in a third-party device. Time-scale offloading types and the generic steps considered in computation offloading are reminded, respectively, in Sections 3.7.4 and 3.7.5. The granularity of offloading is considered in Section 3.7.6. Finally, the framework nature is described in Section 3.7.7. Figure 3.9 depicts these different axes.

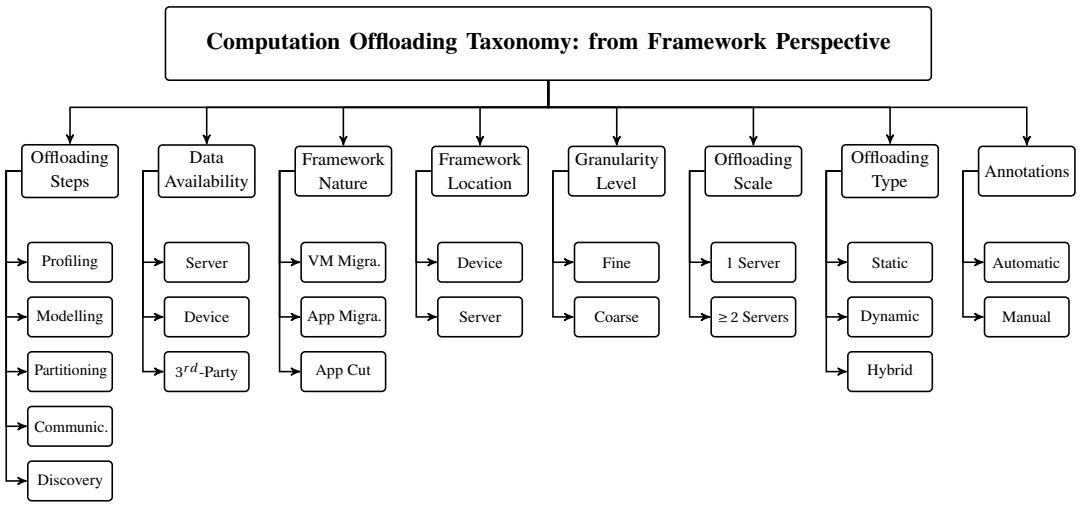


Figure 3.9: **Frameworks-based computation offloading taxonomy**

3.7.1 Code Annotation

Most applications are designed in two separate sections:

- *The desired functionality section*, which expresses the main job of the application in a procedural manner. Only this section can be offloaded.
- *The deployment feasibility section*, which extends the main section with useful code to manage, for example, parallelism and concurrency, and, especially, to interact with the device and mobile user.

The deployment feasibility section cannot be offloaded due to the dependency constraint (see Section 3.4.4). This section of code is annotated before partitioning to add the dependency constraint to the model and limit the search for all of the decision-making possibilities. The annotation is done by adding metadata to the source code. We cite two methods: (i) *manual annotation*, which is done at the time of design by a developer who expends considerable time and efforts annotating the code by examining the scope of his application components [17, 48, 59, 95, 118, 205, 230, 269, 277, 286, 291]; and (ii) *automatic annotation*, in which a profiler performs a deep profiling of the application to create a graph dependency, which is used to identify relevant information about the possibility of offloading [50, 89, 92, 142, 265, 311].

3.7.2 Framework Location

Two possible locations for the offloading frameworks are identified from the computation offloading taxonomy. Offloading frameworks are designed with functionalities that perform the offloading steps.

- *Mobile device.* The offloading framework is located on the mobile device, which reduces network communications between the mobile device and server. However, it induces resource consumption (CPU, battery, and memory) on the mobile device. Several studies have been conducted with the offloading frameworks located on mobile devices [18, 81, 221].
- *Remote server.* The offloading framework is distributed between the mobile device and one or more remote servers. Modules on the mobile device may include profiling, monitoring, and a proxy; while modules on the server may include modelling, partitioning, and scheduling[27]. Despite, the optimization of resource consumption on the mobile device, this solution induces high network communications to periodically send profiling and monitoring information, which *may* increase the execution time [21, 59].

We compare the two solutions in Table 3.5. We highlight the advantages versus disadvantages of each solution.

Table 3.5: **Comparison between the framework locations**

Location	Advantage	Disadvantage
Mobile device	✓ Saves bandwidth by keeping the profiling data locally, ✓ Accepts mobility, ✓ Hosts all modules locally, hence no consistency issue.	✗ Involves local resources consumption, ✗ Needs a long time to make a decision.
Remote server	✓ Saves local computing resources, ✓ Saves energy as decision making is done by the server, ✓ Reduces memory occupation through profiling transfer.	✗ Updates data on server frequently to make the best decision according to changes, ✗ Increases energy consumption due to high network communications, ✗ Restrains mobility, as modelling and partitioning modules should be pre-installed, ✗ Increases execution time due to network latency induced by the profiling transfer, ✗ Makes incorrect decisions as this latter is delayed due to the profile transfer.

[27] Some frameworks include scheduling module to reorder task execution according to the call graph and execution deadlines

3.7.3 Data Availability

Data may represent inputs and outputs, profiling and monitoring statistics, and code sources.

- *Inputs and outputs.* Inputs are entry parameters needed to execute a program; while the outputs are the results obtained from program execution. In a call graph, the outputs of a vertex (e.g., method or class) are the inputs of the successor vertex.
- *Profiling and monitoring data.* This is gathered information about the execution of an application, which includes CPU execution time, energy consumption, and execution call graph.
- Source code. This represents a program's code in the form of files in which the functionalities of the program are described using a programming language, such as *Java*, *C#*, and $C++$.

In state-of-the-art computation offloading, we distinguish three locations for data.

1. *Mobile device.* Data are located on the mobile device [18, 216, 221]. According to the location of the offloading framework and the identified partitions, data might need to be transferred to the server. Considering the partitions, if we suppose we have two tasks A and B, where the B inputs are the A outputs, and A/B represents the local/remote partition, then both the B source code and A outputs are transferred to the server.
2. *Remote server.* Data are present on the server. That is, tasks A and B are pre-installed on the server (i.e., A and B constitute the remote partition) and profiling and monitoring is done on the server using estimations.
3. *Third-party device* A third-party device could be a device located between the mobile device and remote server [275] acting as a dedicated cache or a server on the Internet. The aim is to save information, such as results of execution, profiling history, and application packages, for future use.

Table 3.6 presents a comparison between these three localities.

3.7.4 Offloading Type

According to the moment at which the profiling, modelling, and partitioning steps are taken, we distinguish three types of offloading, namely, static, dynamic, and semi-dynamic (or hybrid) offloading. The reader should refer to Section 2.2.2 for details about these types of offloading.

3.7.5 Offloading Steps

As stated in Section 2.2, computation offloading follows four main steps, namely, profiling, modelling, partitioning, and communication. At the communication step, the server is supposed to be known by the mobile device. According to the literature, we distinguish two approaches for the servers *discovery*; *manual* and *automatic*. In the manual approach, the mobile device already knows the server, for example,

Table 3.6: **Comparison of data availability location**

Approach	Advantage	Disadvantage
On Mobile device	✓It keeps monitoring information, ✓Does not consume bandwidth for profiling transfer.	✗Induces a local overhead due to the profile step, ✗Needs additional resource consumption, ✗Needs to transfer code and inputs to server, ✗Consumes bandwidth and increases the execution time.
On remote server	✓Does not need profiling on mobile device, ✓Does not waste resources for profiling, ✓No latency or transfer for remote code.	✗May make an incorrect decision as profiling is based on predictions, ✗Does not support user mobility due to the pre-installation of tasks.
On third-party machine	✓Keeps history of different executions, ✓Does not waste resources, ✓Supports user mobility,	✗Creates data consistency issues between the three entities, ✗Forwards date to the server, ✗Consumes bandwidth to save data and to forward them to the server, ✗Induces network latency due to data forwarding, ✗Security issues: .Attacker can eavesdrop on the network, .Attacker can change the data state.

a trusted Cloudlet with a static IP address. In this case, communications are manually established. While in the automatic approach, the mobile device seamlessly *discovers* available servers using discovery mechanisms such as *Jini* [13], *Universal Plug and Play (UPnP)* [195], or *Service Location Protocol (SLP)* [282]. If the offloading framework supports collaborative offloading, then servers should discover each other. The discovery step should be fast, especially in a high mobility environment.

The authors in [94] used the concept of *master* and *slave*. The master is the mobile device; while the server represents the slave. The slave runs a JVM to host any offloaded computation. When the mobile device decides to offload a computation, it initiates a discovery protocol to find nearby idle servers, which use a *wireless broadcast* discovery message, that can be adapted to UPnP or Jini. The GnJ framework [93] defines a *service discovery server* used by surrogates to register themselves via Extensible Markup Language (XML) descriptions. Mobile devices discover the surrogates by querying the service discovery server using XML requests. The service discovery server searches for the surrogates that match the mobile device request. The authors in [288] propose a middleware with an hybrid approach that offloads computations or adapts the application configuration to match it with the capabilities of the mobile device. The middleware is built on the *OSGI* service platform, which includes various modules, one of which is the *java Service Location Protocol (jSLP)*. jSLP is a lightweight Java implementation of the SLP defined in RFC 2608[28] for resource discovery.

We propose to compare four well-known service discovery mechanisms in Table 3.7

3.7.6 Offloading Granularity

The granularity of offloading is the degree to which an application is partitioned. Current frameworks implement two granularity levels: *fine-grained* and *coarse-grained*. Figure 3.10, depicts the two granularity levels. The purpose of granularity is to deal with the security, consistency, performance, and

[28] https://tools.ietf.org/html/rfc2608

Table 3.7: **Comparison of service discovery mechanisms**

Mechanism	Description	Advantages versus Disadvantages	Network type	Multicast	Broadcast	Approach
Service Location Protocol [282]	An IETF protocol, centralized and proactive. It includes a mobile agent (a centralized service repository), which is used by a service agent and a user agent, respectively, to register and locate required services.	✓Fast search, ✓Discovering services on a various range of networks, ✓Supports a large number of nodes, ✓Uses caching, authentication, and scoping. ✗Centralized, ✗Has a single point of failure, ✗Vulnerable to DoS attacks.	Fixed	Yes	No	Centralized
Jini [13]	It is an open-source Java program. It uses a similar strategy to SLP, but has additional functionality. Services register their attributes with at least one lookup service. The service is contacted later by the Jini client. The lookup service store both service information and the proxy responsible for code execution, locally or remotely.	✓Services are recognized automatically, ✓Services are available to everyone, ✓Supports a flexible network, ✓An open-source program, ✗May incur large overheads, ✗Needs a reliable, stream-oriented transport protocol (e.g., TCP), and multicast support. ✗Not attractive in a wireless ubiquitous environment, ✗Vulnerable due to the single point of contact.	Fixed	Yes	No	Centralized
IBM's Salutation Protocol [57]	An open standard providing a service discovery protocol and session management protocol. The centralized repository is called a salutation manager. It was primarily designed for home and enterprise environments. Devices, services, and applications advertise their capabilities, and discover and access each other.	✓Support only password-based authentication, ✓It implements two interfaces for applications and the transport layer, ✓Flexible, uses various underlying transport protocols, ✓It maps on Bluetooth service discovery,	Fixed	No	No	Centralized
Universal Plug and Play [195]	Developed by Microsoft and the UPnP Forum. It defines a set of protocols allowing various types of devices to connect seamlessly to the network. Built upon the standard IP, it is a distributed query-based model	✓Decentralized, ✓Robust against communication errors, ✓Provide security mechanisms. ✗Burden the bandwidth-constrained wireless links in ubiquitous environments, ✗Cannot scale well due use of multicasting, ✗It does not support attribute-based querying for services.	Fixed	Yes	No	Centralized

complexity issues summarized in Table 3.8.

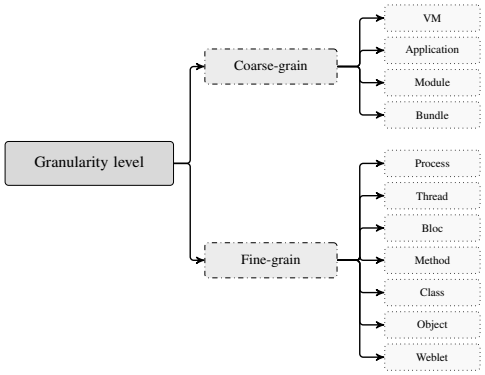


Figure 3.10: **Taxonomy around migration patterns**

1. Fine-granularity includes: thread/process [50, 205], bloc (loop) [92, 290, 291], method [59], class [95, 217], object [209, 277, 296], and weblet granularity [322].
2. Coarse-granularity includes: VM [175, 250, 259], application [77, 304], module [17, 48, 230, 269, 311], and bundle granularity [89, 91, 142]

Table 3.8: **Granularity advantages versus disadvantages**

Strategy	Advantage versus Disadvantages
Fine-grained	✓ Saves energy compared to coarse-grain because only the parts that benefit from remote executions are offloaded, ✓ Increases flexibility since more opportunities are created to locate functionality on remote servers, ✓ Supports high mobility, ✓ Increases security as a thread or method has less meaning to attackers than modules or entire applications. ✗ Intensive monitoring at runtime due to the large number of application components to cover, ✗ Resource intensive synchronization mechanism, ✗ Must ensure consistency between mobile device and server, ✗ Increases the resolution overhead due to the large representation of the application, ✗ Programmer greatly involved in annotating the source code for their applications.
Coarse-grained	✓ Simple offloading mechanism, as it reduces programmer responsibility, ✓ Requires low monitoring overhead as the application is represented as a whole or with only a few components, ✓ Less resources are used (memory, CPU, and energy) in the profiling, modelling, and partitioning steps, ✓ Reduces the resolution overhead as the application is represented with a small model. ✗ Increases data transmission, as large classes cause large migration and remote invocation overhead, ✗ Not suitable for mobility; larger tasks increase the probability of task completion failure due to server disconnection, ✗ Increase the vulnerability to network threats, ✗ Not suitable for resource-constrained environments, such as edge computing.

3.7.7 Framework Nature

This section focuses on the models used to describe the partitioning optimization problem. We classify the state-of-the-art models into three axes. In the first, VM migration is adopted to offload computation; the second axis considers application migration in order to save bandwidth and latency induced by VM migration; the third and last axis describes works that effectively spread applications into a client-server model. While the first two approaches are proposing to migrate the application as a whole, without profiling or partitioning, in the third approach, the application is partitioned into local and remote partitions. Figure 3.11 portrays these different axes with a focus on the partitioning models.

3.7.7.1 VM Migration

The application is hosted on a VM, which is migrated to a remote server for computation. On the server, a VMM reserves a pool of resources for the migrated VM, which is then resumed. When the application execution finishes, the VM is migrated back to the mobile device (mostly laptops). VM migration uses two approaches:

- *Cold migration*. The VM is stopped during the migration step when the entire VM is migrated to the server. The VMM will then resume the VM and run the application. The *downtime*[29] is proportional to the VM size; larger the VM size, the longer the downtime is. Cold migration is used for application running on background, such as virus scanning, indexing files for fast searches, and analysing photos.

[29] Is the period of time during which the VM is not accessible.

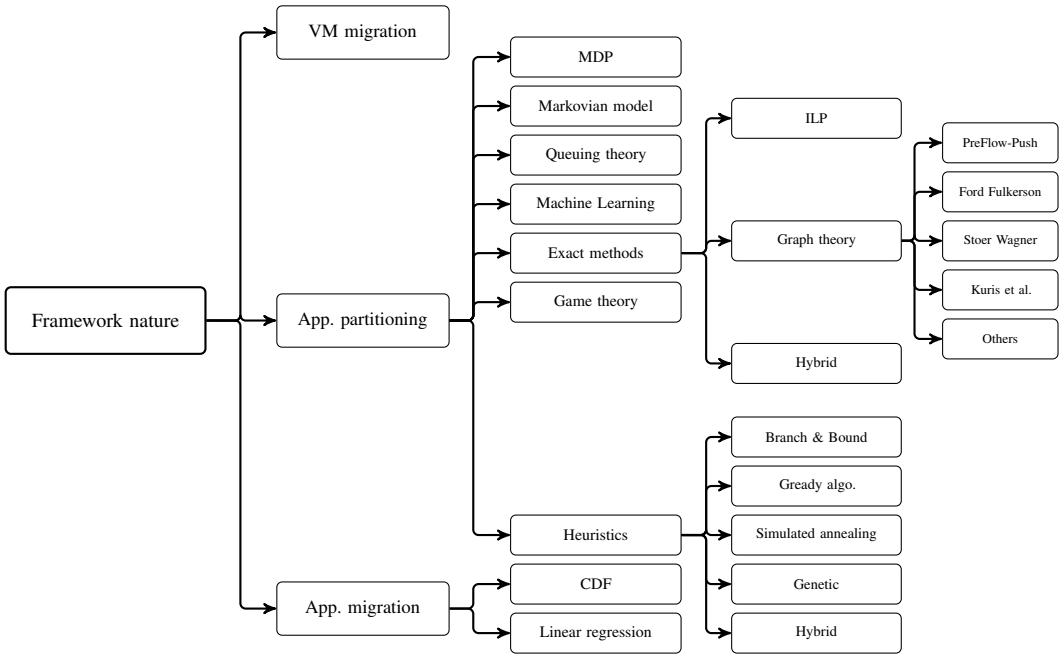


Figure 3.11: **Taxonomy around framework nature (modelling and partitioning algorithms)**

- *Live migration* or hot migration. This minimizes downtime by ensuring execution continuity of services and applications during the migration step. The state of the VM on the server is progressively updated during migration. Live migration is carried on various network protocols, such as the Locator Identifier Separation Protocol (LISP) [232] and IP mobility [299].

To ensure a good QoE, we should consider the case of user mobility in VM migration. Follow me Cloud (FMC) is a recent approach employing live migration [149] which manages VM mobility to optimal location, according to user location. Various VM migration-based frameworks been proposed [139, 175, 184, 248, 250, 259].

3.7.7.2 Entire Application Migration

Entire application migration consists on offloading the entire processing job of the running application to remote servers. In the beginning, the application is running on the mobile device. In critical condition, including, battery draining, insufficient memory, and/or approaching processing deadlines, the running instance of the application is offloaded to a remote server [170]. We distinguished two approaches in the literature:

- *CDF:* The authors in [304] have proposed to use the *timeout* approach [145] to reduce energy consumption on mobile devices. The application is initially executed on the mobile device with a timeout (set to the minimum computation time that can be benefit from offloading). If the computation is not completed by the timeout, then it is offloaded. The timeout is called the *break-even time*. To compute the timeout, the authors proceeded as follows: first, they formulated the energy

consumption on the mobile device for offloading the application using the following equation: $P_a T_c = P_{tx}\frac{D_{in}}{R_{tx}} + P_{rx}\frac{D_{out}}{R_{rx}} + P_l\frac{T_c}{\alpha} + P_n\frac{T_c}{\alpha}$, where P_a/P_l, P_{tx}/P_{rx}, and P_n represent, respectively, the active/idle power of processor, the transmission/reception power, and the idle power of the network interface. T_c/T_s is the execution time on the client/server. D_{in}/D_{out} is the inputs/outputs size. Finally, R_{tx}/R_{rx} represent the transmission/reception speed, and α is the server speed. The authors have then solved the above equation for T_c to obtain the break-even time T_{be}: $T_{be} = \frac{P_{tx}\frac{D_{in}}{R_{tx}} + P_{rx}\frac{D_{out}}{R_{rx}}}{P_a - \frac{P_l}{\alpha} - \frac{P_n}{\alpha}}$.

The authors have stated that the statistical distribution of the execution time is stable and changes slowly. Therefore, they proposed to use a discrete cumulative distributed function (CDF) of the execution time for the past execution instances. The authors have divided the range of execution times between *zero* and the *worst-case* execution time, into n equal intervals t_i, $i = 0, 1, ..., n$. The discrete CDF is denoted with ψ. The probability of consuming the i^{th} interval is: $1 - \psi(i-1)$. To minimize the total energy used by the client, the authors have formulated the problem via Equation 3.24. Here, δ_i represents the interval length, and x denotes the number of intervals for the optimal time-out. x is an integer variable in the set $\{0, 1, ..., n\}$. To find x, and, therefore, the optimal time-out, the authors have calculated the energy consumption for all the values from 0 to n using Equation 3.24. The x value that results in the least energy consumption is then the optimal time-out.

$$Minimize \quad \sum_{i=1}^{x}\left(1-\psi(i-1)\right)\delta_t\left(P_a+P_n\right) + \left(1-\psi(x)\right)\left(P_{tx}\frac{D_{in}}{R_{tx}} + P_{rx}\frac{D_{out}}{R_{rx}}\right) + \left(\left(1-\psi(x)\right)x\delta_t + \sum_{i=x+1}^{n}\delta_t\left(1-\psi(i-1)\right)\right)\frac{P_l+P_n}{\alpha} \tag{3.24}$$

- *Linear Regression:* The authors in [77] proposed to use linear regression to dynamically and seamlessly adapt to the varying execution times through computation offloading. Their approach consists of continually monitoring the execution time t_{core} of an application, then calculating a linear regression to predict the evolution of t_{core} based on the history of executions. The authors have defined a well-known value, the *cycle period*, t_p. For instance, in game engines, the value of t_p is equal to $33.3\,ms$ to generate 30 frames per second (fps), which is a satisfying value for gamers. Therefore, if t_p cannot be achieved in the future, then the computation offloading of the application is triggered in advance. In addition, when computation offloading is no longer advantageous, the application execution again takes place on the mobile device. The decision to offload the application is taken when Inequality 3.25 is true.

$$\left\lfloor \frac{t_{xMaxCap}}{t_p} \right\rfloor \times t_p \leq \left\lceil \frac{t_{mob}}{T_p} \right\rceil \times t_p \tag{3.25}$$

where $t_{xMaxCap}$ is the estimated time at which the t_{core} will exceed the maximum reserved CPU capacity t_{MaxCap} per cycle period t_p. This time represents the intersection between the regression linear line and the horizontal t_{MaxCap} line. $t_{xMaxCap}$ is computed as follows: $t_{xMaxCap} = \frac{t_{MaxCap} - b}{m}$, where m and b are the parameters defining the linear regression line $t'_{core} = mt + b$, m is computed as follows: $m = \frac{n\sum_{i=0}^{n}(t_i \times t_{core,i}) - \sum_{i=0}^{n} t_i \times \sum_{i=0}^{n} t_{core,i}}{n\sum_{i=0}^{n}(t_i)^2 - \left(\sum_{i=0}^{n}\right)^2}$, and b is formulated with $b = \frac{\sum_{i=0}^{n} t_{core,i} - m\sum_{i=0}^{n} t_i}{n}$. n

is the number of past executions. The other value that is required to find when to start offloading is t_{mob}, which is the elapsed time between when the offloading decision is taken and when the application is ready for execution on the remote surrogate.

3.7.7.3 Application Partitioning

The application is represented by a theoretic model, such as graph theory. Then the model is solved using either well-known algorithms or heuristics. The model is solved to satisfy an objective function. Various solutions have been proposed to represent and partition an application. For the sake of accuracy, exact methods have been used to find the optimal partition, we can name graph models with partitioning algorithms, such as Ford Fulkerson, Stoer Wagner, and Kuris et al., are applied to find the minimum cut, and the 0-1 ILP. However, the graph and ILP resolution may take a long time, which drastically increases with the size of the model. To cope with this, heuristics have been proposed: Branch & Bound, Gready, and genetic algorithms, to name a few. In recent years, application modelling has shifted to game theory, MDP models, Queuing theory, and machine learning representation. In this following section, we describe these different approaches.

i. Exact Methods.

- *Linear Programming (LP):* LP, also called linear optimization, is a mathematical representation of a given problem aiming at minimizing or maximizing one or more metrics, which are represented in a linear equation. Formally, an LP is a methodology that searches for an optimal solution of a mathematical problem using a linear objective function, subject to linear equality and inequality constraints. The main benefit of LP is that it always finds the optimal solution, when it exists. However, solving LP problems may take a long time, which increases with the problem size [207]. In the context of computation offloading, LP formulates applications into a mathematical optimization problem, where, for example the objective function is to optimize energy consumption or improve performance. Some or all the variables are restricted to be integers. Three variants of LPs have been implemented for computation offloading namely: *ILP*, *0-1LP*, and *Mixed Integer Linear Programing* [95, 142, 144, 217, 285, 287, 312, 322].
- *Graph:* An application is a collection of interacting processing components. When a component calls another one, the two components interact. In a typical client-server model, the client and server components belong to the same distributed application. The client components are computed locally on the client, and the server components run remotely on the server. Formally, we define A as an entire set of application components and, P_s (P_d, respectively) is the remote (local, respectively) partition. P_s and P_d should satisfy the two constraints: $P_s \cup P_d = A$, and $P_s \cap P_d = \varnothing$. The first constraint stipulates that each component of the application is executed either on the client or on the server, while the second property points out that each component is executed at only one location. The application partitioning problem is equivalent to the graph partitioning problem. An

application can be represented with an oriented graph $G = (V, E)$, where V is the set of n vertices ($n = card(V)$[30]), and E is the set of edges connecting these vertices. The vertices represent the software components of the application, and the edges represent the interactions between components. We associate costs $\langle w_1, w_2, \ldots, w_k \rangle$ ($k \in \mathbb{N}^*$) with each vertex $v \in V$ that represent the parameter or context of the component (usually execution time or energy consumption). An edge $e_{ij} \in E$ is associated with the frequency of invocation and data access between the two vertices v_i and v_j. The granularity level chosen to represent an application impacts highly the graph size and its complexity.

The decision on where to place a component, on the server or on the mobile device, depends highly on the objective function. This optimization problem is equivalent to finding a minimum cut for the corresponding graph. A minimum cut of a graph is a partition of the graph vertices into two disjoint subsets that are joined by at least one edge. The cut is minimal in the sense of considering the values affected to the graph. Several past studies have used graph partitioning to make offloading decision, we cite [5, 6, 90, 91, 95, 209, 217, 223, 269, 286, 291].

- *Hybrid:* Some frameworks incorporate the combined features of the graph-based and LP-based application partitioning algorithms (APAs). They tend to extract the important features of graph-based and LP-based APAs to improve application performance. The first step is to represent the application using a graph, then an ILP is drawn, based on the graph representation. Some studies have been done using the hybrid approach, these include [50, 59, 92, 205, 266, 312].

ii. Heuristics.

- *Genetic Algorithms (GA):* The GA is an heuristic that belongs to the class of evolutionary algorithms. It is a population-based method, inspired by the process of natural selection, which aims to evolve toward a global optimized solution from a population of solutions [61]. Each candidate solution corresponds to a *chromosome*, which can be mutated and altered, in the population of *genes*. A typical genetic algorithm requires: (i) a genetic representation of the solution, i.e, *encoding*; and (ii) a *fitness function* to evaluate the solution domain. A random solution is chosen by the algorithm, then improved through a repetitive application of *selection*, *crossover*, and *mutation*. In the initialization phase, a set of parameters should be defined, namely, the population size N, the maximum iteration number I, the mutation probability p_m, and the crossover probability p_c. In the selection phase, chromosomes are selected to recombine in order to generate the next population via crossover and mutation. In the crossover operation, two parent chromosomes which will generate a high-quality offspring (chromosomes) are combined. The mutation phase softly modifies the chromosomes to improve their fitness and avoid early convergence.

  In a computation offloading optimization-based problem, a population represents the offloading strategy encoded as a vector of n integers $\langle x_1, x_2, ..., x_n \rangle$, where n is the total number of components

[30] $card(V)$ for cardinal, is the total number of elements in the set V.

in the application. Shuiguang et al. [64] have used GA to solve the offloading optimization problem. They used the objective function $F(m) = w_m \times L(rt_s^F) + (1 - w_m) \times (\sum_l vE_l + \sum_c E_c^F)$ to calculate the fitness value of each chromosome. m represents a given mobile device; w_m is a weight coefficient set according to the status of the mobile device; $L(rt_s^F)$ is the overall, final execution time for processing a mobile service on both the mobile device and the Cloud; E_l is the energy consumption of a locally executed service; and E_c^F represents the final energy consumption of the service c. At the initialization, each chromosome $X^i = (x_1^i, x_2^i, ..., x_d^i)$ is a population $i \in \{1, 2, ..., N\}$; d is the number of services in the mobile service workflow; x_j^i indicates whether the j^{th} service in the generation X^i should be offloaded (i.e., $x_j^i = 1$) or computed locally (i.e., $x_j^i = 0$). The probability to select a chromosome for recombination is related to its fitness value F_i, which is given by $pr_i^s = 1 - \frac{F_i}{\sum_{j=1}^{S} F_j}$. In the crossover operation, the selection of the parents, which generate the offspring is based on the selection probability pr_i^s. To this aim, the authors first, compare the local fitness of each gene by calculating a weighted factor $f_i = w_m \times rt_i^F + (1 - w_m) \times E_i^F$, where rt_i^F and E_i^F represent, respectively, the final execution time and energy consumption of a service i computed considering the mobility and fault-tolerance. Once the fitness of each gene is calculated, the genes with the best local fitness are chosen and copied to the offspring. In the mutation phase, the gene selection probability is $pr_g^m = 1 - \frac{f_g}{\sum_{j=1}^{d} f_j}$.

- *Queuing Theory:* This is a branch of mathematics that studies and models the act of waiting in lines.The queuing theory is characterized by a set of parameters, namely: (i) *the arrival process of customers*, which are assumed to be independent and have a common distribution; (ii) *the service times*, which are independent and identically distributed; (iii) *the service discipline*, in which customers can be served using the First In First Out (FIFO) model, random order, or with priorities; (iv) *the server capacity*, which is the number of servers computing in parallel; and (v) *the waiting room*, which is the queue capacity. Kendall has introduced a notation, under the form $a/b/c$, characterizing the range of these queuing models. Here, a specifies the arrival time distribution, b is the service time distribution, and c is the server capacity. For instance, models $M/M/1$, $M/M/S$, $M/G/1$, $G/M/1$, $M/D/1$, and $M/M/1/N$ have been used in various scenarios. $M/G/D$ represent the exponential/general/deterministic distribution, while S/N describes the server/queue capacity. Rong et al. [240] proposed to use queuing theory, combined with a Markovian Decision Process (MDP), to extend the lifetime of battery-powered mobile devices through computation offloading. The proposed framework is composed of three components; the client (mobile device), the server (base station), and the wireless channel. The authors have modelled the client as a Continuous-Time Markov Decision Process and proposed an interaction model for the client and remote server. The wireless channel has been modelled with a Markov chain using two states, in which each state is assigned a constant Packet Error Rate PER. The server is an infinite $M/M/1$ queue. The authors have defined two policies to make the optimal decision, the main difference is whether or not to consider the wireless channel and server stable. These policies are the *offline*

optimal policy, which is based on the joint stochastic model using LP, and the *online optimal* policy, in which an $M \times N$ matrix is constructed, where each entry (i, j) corresponds to an offline optimal solution obtained for given values of PER_i and $P_{reject,j}$ (probability of rejection).

- *Markov Decision Process (MDP):* The MDP is a mathematical model of a problem in which decision making occurs in situations where the outcomes are partly random and partly under the control of the user. MDP is comprised of a decision agent, which checks the current state s and repeatedly makes a decision to perform an action a with a probability P. This transits it to another state s' and generates a reward r. An MDP is defined by the fourfold $\langle S, A, R, P \rangle$, where S defines all the possible states of the system which are known to the decision-maker (user), A encloses all the possible actions that can be taken by the decision-maker, R represents the reward for taking an action a in the state s, and finally, P represents the transition probability. The aim behind using MDP is to find the optimal action to take for all possible combinations. Nasseri et al. [203] have proposed an MDP-based approach combined with look-up tables for collaborative processing in an ad-hoc network. The authors have used the notion of reward and credit between the helpers and requester. Each mobile device is classified into a matrix according to its energy potential. A helper can accept or reject a request according to a calculated reward. The authors have used an MDP algorithm to calculate rewards. In the calculation of the reward, two parameters are considered, power and delay. The power reward function is given by: $f_p(s, a) = \frac{1}{1+e_a^p}$, where p_a denotes the power consumption of the current state for a given action a taken at a time t. Similarly, the delay reward function is expressed by: $f_d(s, a) = \frac{1}{1+e_a^d}$, where d_a is the response time. The sum of the delay and energy reward functions with a weight factor is given by: $f(s, a) = w_p \times f_p(s, a) + w_d \times f_d(s, a)$. Therefore, the total reward function is $r(s, a) = f(s, a) - h(s, a)$, where $h(s, a)$ is the transition cost function.
- *Game Theory:* Game theory [194] studies mathematical models of conflict/cooperation between players, which are competing to use a set of resources. Each player chooses a set of resources to maximize/minimize his utility or cost. The cost and utility depend on the number of players in competition. a congested game is defined by a set of parameters $\langle N, R, \sum_i, n_j, S_{ij} \rangle$, where $N = \{1, ..., N\}$ is a finite set of n players; $R = 1, ..r$ is a finite set of resources; $\sum_i$ is the player i's set of strategies; n_j is the number of players who chose the strategy j; and finaly, S_{ij} is the cost function for player i when selecting resource j. Algorithms are defined to search for the *Nash equilibrium*, which is the optimal policy where no player can benefit by changing strategy while the other players keep their strategies unchanged. Ge et al. [86] used game theory to optimize the overall energy consumption in the MCC system. The energy optimization problem was formulated as a congestion problem between various players (mobile devices). Each player's strategy was to select the server minimizing overall energy consumption. The authors have shown that a Nash equilibrium always exists. Other studies include [34, 42].
- *Machine Learning:* Machine learning explores the study and construction of algorithms that can

learn from and make predictions on data. Machine learning appears in many guises, including web page ranking (Google engine), collaborative filtering (Amazone, Netflix), and automatic translation of documents. Machine learning appears in several forms, including; supervised/unsupervised, active/passive, helpfulness of the teacher, and online/batch learning. *MALMOS* [71] is a machine-learning-based mobile offloading scheduler, which makes a decision to offload a computation or not. MALMOS provides an online training mechanism to dynamically adapt decision making based on previous decisions. Other studies using machine learning to make offloading decisions have been proposed in the literature, we cite [16, 72, 83, 261].

3.8 Conclusion

A comprehensive survey of computation offloading in the literature, along with classifications, has been presented in this chapter. For each actor impacting the performance of computation offloading, we proposed a survey that is in line with computation offloading. A comparison between approaches has been provided. A review and comparison between the frequently used offloading frameworks in state-of-the-art computation offloading was also described in this chapter using different points of view. Although several works have been proposed to remedy the problem of resource poverty and scarcity in mobile devices, a great effort is still required to accommodate applications, such as 3D-gaming, VR, and AR on current mobile devices. Indeed, these types of applications are of special interest, due to their real-time constraints, high resource demands, and complexity. Most of the old frameworks reviewed in this manuscript are obsolete, and they cannot manage such constraints. In the next chapter, we propose to dissect one of the most complex applications on the market, game engines, which is also a computationally intensive, resource demanding, and real-time interacting application.

CHAPTER 4

PERFORMANCE ANALYSIS OF GAME ENGINES ON MOBILE AND FIXED DEVICES

4.1 Introduction

Game engines, like Unity 3D [1], are well-established tools to generate interactive, fully-multimedia environments that range from games to serious health applications. Game engines are complex software, which consume a lot of resources (including CPU, GPU, energy, and memory) to meet the demand of gamers. Indeed, the QoE depends on the interactivity and on the multimedia quality. Therefore, game developers design their best-seller games so that the game engine exploits the dedicated hardware that is typically present in “boxes" (e.g. Xbox, PS4), however resource-limited devices such as mobile devices (smartphone and tablet) cannot run these games, or at a much lower quality. Game engine developers are looking for solutions to address this issue and to embrace the mobile market, which represents billions of dollars of revenues.

To get rid of the limitations of end-user devices, one of the envisioned solutions is to leverage Cloud computing infrastructure. Developers of games have three options: *Computation offloading*, *Cloud gaming*, and *Client-server architecture*. Whilst the third approach (client/server) is widely implemented and mastered (research work mostly addresses network management [76, 234, 276]), the two first approaches need more investigations regarding the software aspect of the game engines. Computation offloading requires low latency communication with the remote servers, which might be solved using MEC [47, 126] and Fog Computing [22, 157]. However, deciding which module or function should be offloaded remains an open issue. The latency issues of Cloud gaming have also been frequently studied [56, 121, 233, 243]. But, the management of virtualized resources (known as *server consolidation*) to prevent performance degradation and resource congestion has been rarely addressed [108, 325].

To identify the best implementation option, developers should take into account the nature of the game and the considered platform. However, the body of literature related to performance evaluation for

game engines is surprisingly small with regard to the significance of the gaming industries. In this regard, we may mention two position papers that called for research on game engine architecture [10, 281], and some other work that focused on applying game engines to specific areas such as serious games [58], research [163], and VR serious applications [58, 141, 185]. The literature is missing work on the architecture and the performance analysis of modern engines. To fill this gap, we analyze in this chapter the performance of a well-known engine, Unity 3D.

4.2 Background: Game Engines

Game engine can be defined either as a framework for game creators or as a piece of code for gamers. Accordingly, two definitions arise:

- The game engine is the set of tools (*i.e.*, including low-level libraries, UI editors, and game multimedia management tools) that facilitate the work of a game developer in the process of creating a new game. The community of game developers considers thus a game engine as a framework or a platform. The framework provides an abstraction layer between the game content (*i.e.*, multimedia content and main scripts) and the underlying hardware.
- The game engine is the set of software and data that runs on a device to provide the game to an end-user. The community of gamers considers a game engine as a piece of code. The border between the game and its framework is often confused. Some frameworks offer a clear separation, while others do not. All games created by Unity 3D share similarities, making them consistent Unity 3D game engines. We focus in this chapter on typical Unity 3D software, which we refer to as a game engine.

The delay for the result of an action to appear in the screen is central in gaming since the QoE is linked to interactivity [121]. We distinguish different *genres* of games with different requirement. An action game can be a First Person Shooter (FPS) or a Third Person Shooter (TPS) depending on whether the player is immersed in the scene or the avatar representing the player is visible. Studies [55, 160, 229] have shown that the acceptable delay depends on the game genre and varies from 100 to 200 ms, and even up to 500 ms for Role-Playing Game (RPG) and Massively Multi-player Online Games (MMOG). Given their specific constraints regarding short delays, the management of FPS has received scientific efforts [12, 165]. In this chapter, we tested several game genres and considered these requirements.

Another key criteria for high QoE is the frame rate. Gamers get the feeling of immersion in an animated world when the game engine generates a high number of Frames Per Second (fps). Less than 30 fps is widely seen as non-tolerable [54] and the recent trends in video and interactive multimedia is to deliver frame rate up to 90 fps [201]. The *rendering pipeline* (see Section 4.2.3) is the part of the game engine that generates one frame every x ms (x ranges from 33 to 10). A large part of our measurement study is on the time needed to generate one frame.

4.2.1 Game Architectures

We focus now on the game architectures. Figure 4.1 depicts the characteristics of the discussed architectures by indicating the location of the various modules of the game engine and the used mechanisms to communicate and exchange data or commands.

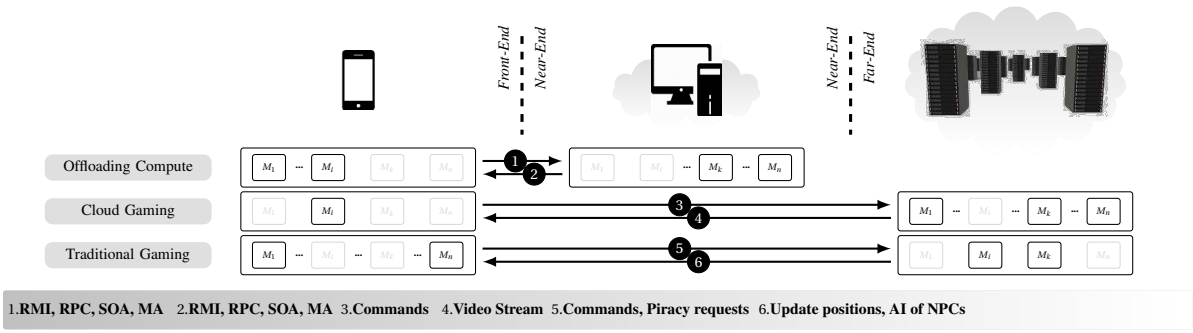


Figure 4.1: **Comparison of gaming architectures**

Traditional client-server. The game engine runs almost entirely on the client side and the server assists the game engine, either as a secured session manager or as a multi-player gaming enabler. When the server is a session manager, it manages authentication, updates game assets (for example new levels and new graphical environments) and matchs fighting players [289]. As a multi-player gaming enabler, the server takes a stronger role: **(i)** it gets all the gamer input commands and forwards them to the other players, **(ii)** it implements anti-cheaters policy and acts as referee [306], and **(iii)** it manages Non-Player Character (NPC). Several architectures exist to deal with scalability. (1) *client-server* architecture with a single server [268], (2) *mirrored client-server* [206,300,301], and (3) *peer-to-peer* architecture [30,307].

Cloud gaming. The entire game engine runs in a remote server. The client includes only UI command and a video decoder. This architecture is similar as an interactive video delivery system. The player sends the commands to the server. The engine first converts the client commands into appropriate in-game actions, then computes the game logic, and renders the game scene. This scene is then compressed by a video encoder and forwarded to a video streaming module, which finally streams it to the client. The client decodes the video and displays the frames [49,111,174].

Cloud gaming solution relies on the concept of *resource consolidation*, which consists of using the virtualization technology: each game engine is encapsulated within a virtual machine (or a container) and dynamically mapped onto a pool of physical resources including CPU, GPU, memory, and Inputs/Outputs (I/O). Hence, these game engines concurrently share the server resources through the hypervisor. Studies have shown that virtualization techniques barely degrade the performance compared to a configuration where the game engine is the only program running in a bare-metal [256].

Two main challenges arise with the Cloud gaming architecture: the delay and the resource use. In comparison to the delay issue in the traditional client-server architecture, Cloud gaming adds extra-delay due to video encoding and decoding. Despite various proposals [171,254], both processes take around

30 ms [257]. With respect to the network communication delays (around 50 ms in average [46]), the overall delay is close to the threshold at which QoE is reported to degrade [56, 121]. Regarding resource use, the Cloud Gaming Providers (CGPs) have to find a trade-off. The smaller is the number of games concurrently running on a server, the higher can be the quality setting, but the higher are also the cost. Moreover, the CGPs have to reserve some free spare resources in servers, otherwise some games may interrupt due to concurrent workload peaks. One of our motivations is to get a better understanding of the load variability and the internal modular structure of the game engine so as to make server consolidation more efficient and less risky.

Computation offloading. A subset of the game engine modules run on nearby server(s) [7, 133, 251]. Depending on specific criteria, including network bandwidth, latency, and processing time, the engine is spread into client and server partitions. The server hosts the modules that improve the responsiveness of the game, and the client hosts the remaining modules, typically those interacting with the device components or with the gamer (like UI). The two sides exchange data during the execution. Whenever needed, the server can call a module, or download it using mechanisms such as RPC, RMI, SOA, and MA. The partitioning requires profiling the whole game.

The computation offloading solution presents two constraints. First, as in Cloud gaming, computation offloading requires low latency. Second, the game engines should tolerate module partitioning. One of the advantages is that module partitioning allows a better exploitation of resources, especially by distinguishing the cutting-edge modules that require specific GPU hardware from the standard ones that can accommodate any CPU configuration. Remote graphic rendering is not a new topic [115], but none of the existing solutions match the expectations for fast rendering high-definition, complex, 3D images. Moreover, to the best of our knowledge, no previous work has addressed the case of offloading non-graphic components of a game engine, like physics module, audio, and object behaviour scripts. One of our motivations is to explore offloading different non-graphic components of a game engine.

4.2.2 Game Engine Main Modules

A game engine consists of various modules depending on the game genre. An engine designed for a two-person fighting game is different from an FPS engine, an MMOG engine, a Real-Time Strategy (RTS) engine, or racing engine. Nevertheless, we identify some families of module that are common to most of the game engines. Figure 4.2 shows a baseline architecture of a generic game engine with the main module families.

Some of the module families of the engine are written by the game creator including:

Artificial Intelligence (AI) – these modules emulate an artificial and intelligent behaviour of NPCs, for example, learn and interact socially, exhibit emotions, and even the ability to hunt and the instinct to survive. Different features are used to implement the AI such as: **(i)** *decision-making*, to affect the NPC; **(ii)** *basic perception*; the AI needs some way of perceiving its environment using human-like sensors (including sight, hear, smell, and touch), and navigating using basic mechanisms like

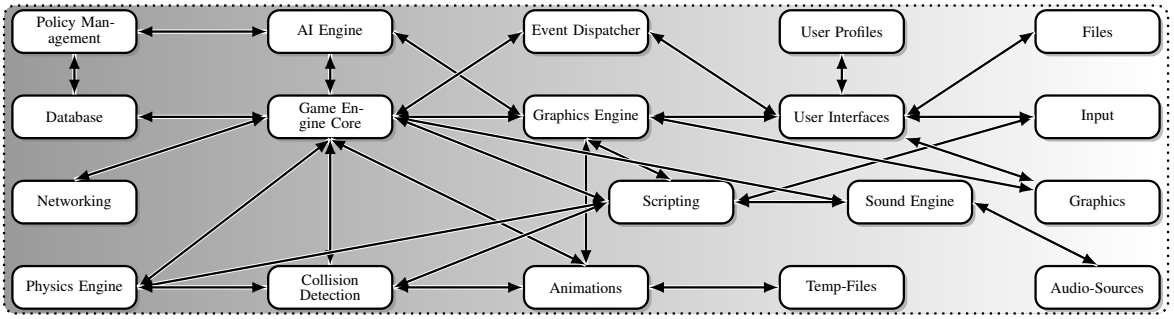


Figure 4.2: **Game engines baseline architecture**

Crash and Turn, and *pathfinding*; **(iii)** *prediction*; the ability to effectively anticipate the player behaviour through historical decisions, random guess or backtracking. ALive[1], GAIGE[2], and kismet[3] are three advanced AI modules.

Scripting – these modules contain the gameplay itself. From the captured inputs obtained from the Human Interface Devices (HIDs), the game developer details the series of game content and events in a scripting language, which is specific to every framework. The scripts can be used to create graphical effects, control the physical behaviour of objects or implement an AI system for NPCs.

Animation – any character in a game (whether it is a human, an animal, or a robot) needs animations (*e.g.* move, jump, stand up, and set down). Five basic animations exist; **(i)** *rigid body hierarchy animation*, **(ii)** *skeletal animation*, **(iii)** *sprite/texture animation*, **(iv)** *vertex animation*, and **(v)** *morph targets*. In particular, skeletal animation generates a pose for every bone in the skeleton and passes them to the rendering engine as a set of matrices that are transformed into a final blended vertex position. This process is called *skinning*. It also interacts with the physics engine to simulate *rag dolls*, which are dead animation characters, whose bodily motion is simulated by the physics.

Some module families are common between most games created within a given framework. These modules represent the abstraction layer and prevent game creators from spending time on low-level issues. In particular:

Physics Engine – it includes collision detection and rigid body dynamics. This module aims to make the game world as realistic as possible using the physics laws. Without this module, objects would interpenetrate, leading to block interactions with the virtual world. This module is integrated as a

[1] `http://alive.sourceforge.net/`
[2] `http://game-ai.gatech.edu/`
[3] `http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/`

third-party SDK. Existing physics engines in the market include *Havok*[4], *PhysX*[5], *Open Dynamics Engine (ODE)*[6], and *I-Collide*[7].

Input – a player interacts with the game using HIDs, *e.g.*, joystick, keyboard, mouse, steering wheels, dance pads, and VR sensors. The input module provides a mechanism to customize the mapping between the physical control and logical game functions. It may include a system for detecting chords, sequences, and gestures. These commands are encapsulated as objects and forwarded to the system.

Networking – this set of routines and protocols enables interactions with a server to set up multiple players and share a sense of space, time, and presence through avatars. In MMOGs, the players join the game dynamically through *game sessions* and interact by exchanging commands [199]. The architecture of most of the MMOGs is based on client/server.

Multimedia Rendering – these modules generate the graphical and audio elements of the game. Graphics are complex aspects of a game. 3D games are built over 3D assets created by a design application like *Maya*, *XNA*, *Blender*, and *WPF*. These assets are mixed with other objects like materials, shadows, lights, and animations to create a realistic scene. This engine is designed as a layered architecture, including: **(i)** *Low-Level Renderer*, where the focus is on rendering as quickly and richly as possible a set of geometric primitives; **(ii)** *Visual Effects*, like particle systems, light mapping, dynamic shadows, and colour correction; **(iii)** *Front End* where 2D graphics overlaid on 3D scene such as in-game menu, a console, and in-game Graphical User Interface (GUI). Regarding sounds, the audio engines vary in sophistication. The audio clips are exported in different formats like mono, stereo, wave files (.wav) or ADPCM files (.vag). Existing commercial audio engines include *Quake* audio engine, *XACT*, *SoundR!OT*, and *Scream*.

4.2.3 Rendering Pipelines

We describe now the rendering pipeline under Unity 3D. We first describe the pipeline that achieves a 3D rendering, and then, we show how Unity 3D implements it.

A rendering pipeline is a combination of successive stages to generate a 2D image for a geometric description of a 3D scene and a virtual camera. Unity 3D is cross-platform: a given game can run on various operating systems and hardware. For each configuration, the engine uses the default rendering pipeline developed for these targets.

- *Direct 3D* is a Microsoft's 3D graphics low-level API. It is used to draw lines, triangles, and points or to launch parallel processing on the GPU. It is the primary competitor of OpenGL. It is used on Microsoft platforms (Xbox 360, Xbox One, and the Windows operating system platforms).

[4] `http://www.havok.com/physics/`
[5] `https://developer.nvidia.com/gameworks-physx-overview`
[6] `httph//www.ode.org`
[7] `http://www.cs.unc.edu/~geom/I_COLLIDE/index.html`

- *OpenGL* is a *Khronos*-developed graphic library. It is the most used API in industry. It is developed to launch a large number of applications on different computer platforms, such as gaming, manufacturing, medical, VR, content creation, and Dynamic Audio-Visual Communication (DAC).
- *OpenGL ES* is a fork of OpenGL. It has been developed for embedded systems, including smart phones, consoles, appliances, and vehicles.
- *WebGL* is a free low-level 3D graphics API based on OpenGL ES 2.0. It is integrated to HTML5 as Document Object Model interfaces. Major browsers (such as Safari, Chrome, Firefox, and Opera) implement this pipeline as a 3D plugin.

Figure A.1 in the appendix, describes the graphic pipelines used by Unity 3D (*i.e.*, OpenGL, OpenGL-ES 2.0, WebGL, and DirectX). Each graphic pipelines is represented by different stages along with their interactions. Each pipeline differs by the number of stages and internal flow, but the general idea of rendering remains the same.

Firstly, a series of geometric operations are done to render a collection of geometric primitives as fast as possible. The data (vertices and indices) of 3D objects are approximated by triangle meshes, forming the basic building blocks. The more triangles are used to approximate an object, the better is the approximation but the more processing power is needed. Each object is then centered at the origin of a local coordinate system with local orientation and size. Thereafter, all objects are brought together in a global coordinate system by applying geometric transformations. At the end, the virtual camera in the scene is translated to the origin of the world space.

Secondly, a set of aesthetic effects (lights, materials, shaders, and textures) is applied to the scene. The *lighting* is a key step to produce a realistic scene. The light sources are simple objects defined in the world space with a combination of colour, intensity, direction, focus, and position. This step also includes a repetition of absorbing and reflecting light processes depending on various parameters (*e.g.*, smoothness and material of the surface and incidence angle).

Thirdly, a set of culling operations (*scene graph/culling optimization*) is done. Each object has two sides with respect to camera position: a front and a back side. The front (respect. back) sides are polygons with vertexes ordered in a clockwise (respect. counter clockwise). All the back-face polygons are culled. The culling step is triggered to also decide which object should be discarded from the scene, according to the computation of the *view frustum*. An object inside (respect. outside) the frustum is kept in (respect. completely culled from) the scene, while the object that is between the inside and outside of the frustum is partially clipped.

Finally, a perspective projection is done to render the 3D vertices into a 2D projection window inside the frustum. The vertex coordinates are transformed to place the 2D scene into a rectangular window on the screen, called the *viewport*. The outputs of this stage are pixels. The *rasterization* transforms these pixels into screen coordinates forming a list of triangles that should be checked and coloured.

Unity 3D uses four additional rendering activities, which occur in conjunction with the main pipeline. We summarize these activities in the appendix Section A.2.1.

4.3 Methodology and Testbed

We describe now the configuration of the devices that we used in our experiment, the process of running games on these devices, a description of the used games, and our measurement.

4.3.1 Platforms

To evaluate the performance of the game engines generated by the framework Unity 3D, we installed Unity 3D v5.03 on top of three devices: a Dell Precision M4800 laptop, a Microsoft Surface Pro tablet, and a Dell PC tower. The latter is used as a server in our second experiment (implementation of Cloud gaming and computation offloading architectures). We used different configurations as shown in Table 4.1. We then used the installed frameworks to compile the games and generate the adequate files for a range of OSs: an ".exe" file in Windows 7/8/10, an ".deb" in Linux, an HTML page with javascript files for web players, and an ".apk" for Android devices. Accordingly, we were able to run the games generated by Unity 3D on a wider range of configurations: not only the said Dell M4800 and Surface Pro, but also a smartphone HTC One (M8) and a Samsung galaxy S6 Edge. Moreover, we were able to implement Cloud gaming and computation offloading architectures.

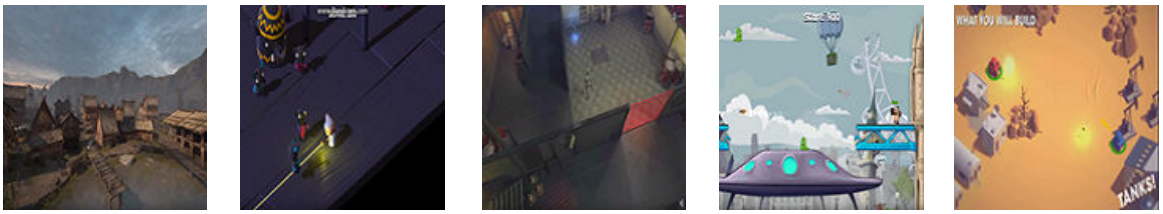
Table 4.1: **Main characteristics of the used platforms**

Platform	Target	CPU	GPU	RAM	Pipeline
Windows 7 Pro Ubuntu 14.04 Mozilla Firefox 44.0.2 Opera 35.0.2066.37	Dell Precision M4800	Intel Core i7, 2.8GHz	Nvidia Quadro k2100M, 2GB	16GB	D3D11 OpenGL WebGL WebGL
Windows 10	Microsoft Surface Pro	Intel Core i5 - 3317U, 1.7GHz	Intel HD 4000	4GB	D3D11
Android 4.4.2	HTC one (M8)	Quad-Core 801 Snapdragon, 2.3GHz	Adreno 330	2GB	OpenGL ES 2.0
Android 5.1.1 Lollipop	Samsung Galaxy S6 edge	Octa-Core Exynos 7420 - 2.1 GHz	ARM Mali T760	3GB	OpenGL ES 2.0
Windows 8.1 Pro	Dell PC tower	Intel Core i7, 3.4 GHz	3x NVIDIA GeForce GTX 780 Ti, 3GB	16GB	D3D11

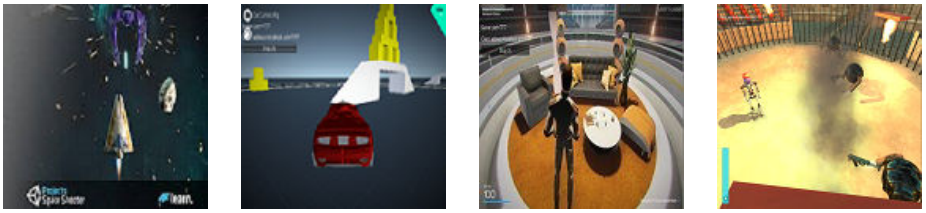
4.3.2 Games

We selected nine games from the "Asset Store" of Unity 3D for our evaluation. These games have different characteristics, which cover the most representative games in the market. A description of the different games is presented in the appendix. Figure 4.3 represents a screen-shot of the different games. We summarize the main characteristics of each game in Table 4.2. For more details about these games, the reader may refer to the Unity 3D asset store.

[8]https://www.youtube.com/watch?v=iV-224nMwN8
[9]https://www.youtube.com/watch?v=iTwtoOO7wXc
[10]https://www.youtube.com/watch?v=LBbPwMmpqQI



(a) Viking Village (b) Survival Shooter (c) Stealth (d) Tower Bridge (e) Tanks



(f) Space Shooter (g) Car (h) Unity Lab (i) Multiplayer FPS

Figure 4.3: **Screen-shot of the different games**

Table 4.2: **Main characteristics of the tested games**

Games	# of players	Dimension	Type	Rendering	Physics	Scripts
Viking Village[8]	1 player	3D	FPS	++++	+	+++
Tower Bridge[9]	1 player	2D	TPS	+	+++	+
Stealth[10]	1 player	3D	TPS	+++	+++	+++
Survival Shooter[11]	1 player	3D	TPS	++	++	++
Tanks[12]	2 players	2D	MMOG	+	++	++
Space Shooter[13]	1 player	2D	TPS	++	+	++
Car[14]	1 player	3D	Racing	++	++	+++
Unity Lab[15]	1 player	3D	TPS	++++	+++	+++
Multiplayer Shooter[16]	multiplayer	3D	MMOG-FPS	+++	+++	+++

4.3.3 Game qualities

For each game, we generated more than 10,000 frames for two quality types: good quality with a reasonable framerate, *i.e.*, around 30 fps, and fast quality, where the game pace is maximum. The engine achieves these two qualities depending on many parameters, which are outlined in Table 4.3.

4.3.4 Measurement

We describe now how we obtain the internal flow and the execution time per frame and per module for the different games over the different platforms and targets, including Cloud gaming and computation offloading architectures.

[11] https://www.youtube.com/watch?v=uRsspkum8LI
[12] https://www.youtube.com/watch?v=paLLfWd2k5A
[13] https://www.youtube.com/watch?v=kX0hnOS1QQQ&list=PLX2vGYjWbI0RibPF7vixmr4x8ONJX-mNd
[14] https://www.youtube.com/watch?v=-Lkbo9ZyYbo
[15] https://www.youtube.com/watch?v=XjiBDKrCsVI
[16] https://www.youtube.com/watch?v=UK57qdq_lak

Table 4.3: **Good quality versus fast quality**

	Quality Settings	Good Quality	Fast Quality
Rendering	Resolution	Full-screen	Full-screen
	Pixel Light Count	4	0
	Texture Quality	Full Res	Eighth Res
	Anisotropic Textures	Per Texture	Disabled
	Anti Aliasing	4x Multi Sampling	Disabled
	Soft Particles	Activated	Disabled
	Realtime Reflection	Activated	Disabled
	Billboards Face Camera Position Probes	Activated	Disabled
Shadows	Shadows	Hard Shadows Only	Disable Shadows
	Shadow Resolution	Medium Resolution	Low Resolution
	Shadow Projection	Stable Fit	Stable Fit
	Shadow Distance	50	15
	Shadow Near Plane Offset	2	2
	Shadow Cascades	Two Cascades	No Cascades
	Cascade Splits	(33,3%, 66.7%)	0
Others	Blend Weights	2 Bones	1 Bone
	V Sync Count	Every V Blank	Don't Sync
	Lod Bias	1	0.3
	Maximum LOD Level	0	0
	Particle Raycast Budget	256	4

Execution Time. The CPU/GPU execution time per frame, for the aforementioned modules, is obtained for 10,000 frames for each quality encoding. We used the Unity Profiler and script to obtain these values.

Internal Flow. We used two tools with a script to identify the Unity 3D internal flow. We dumped the memory using the script that identifies all the classes/objects and methods that are called per frame. We used Visual Studio 2015 profiler[17] and Dependency Walker[18] to check the validity of the obtained results.

4.4 Performance Analysis and Game Classification

Now, we present the results of our measurement campaign regarding the time needed to generate one game frame. We then propose a classification of the games with respect to the best approach to adopt for efficient implementation (i.e., whether the game engine should run on the device, in a Cloud gaming system, or with computation offloading). In Section 4.5, we will study more precisely the case of computation offloading by a thorough analysis at the modular scale.

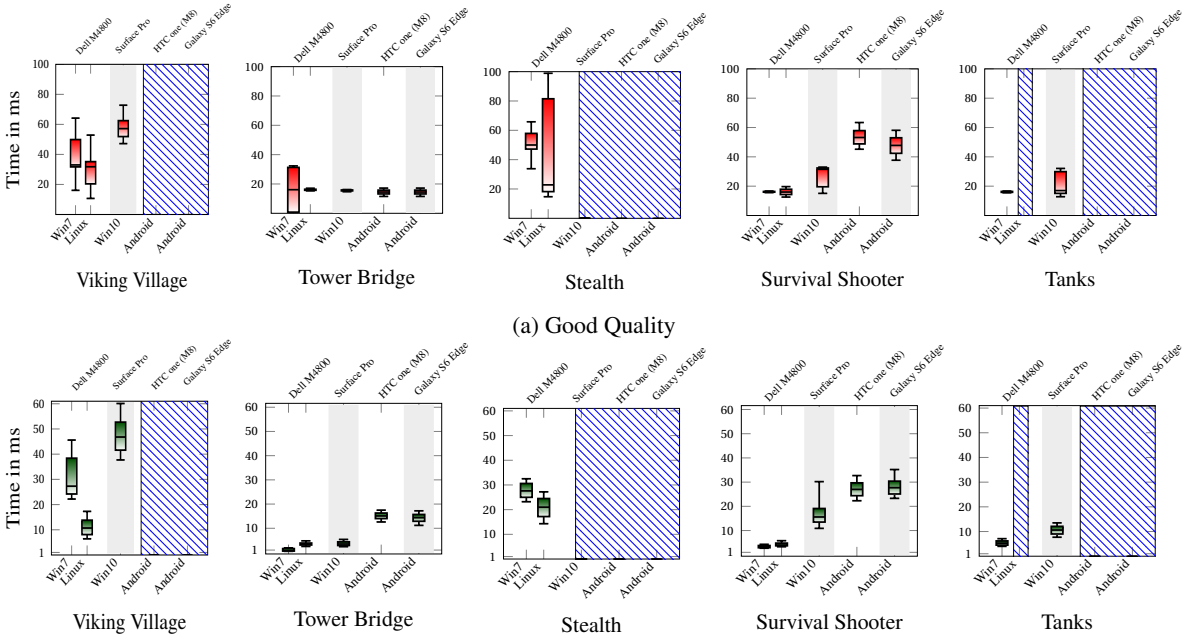
4.4.1 Time Needed to Generate One frame

Figure 4.4 shows the time (in *ms*) spent by the CPU to generate one frame for five games on five platforms. The results in Figure 4.4a (Figure 4.4b, respectively) correspond to the good (fast, respectively) quality. Figure 4.5 represents also the time spent by the CPU to generate one frame for Stealth, Space Shooter, Car, Unity Lab, and Multiplayer FPS games on the Dell and mobile devices for the good (Figure 4.5a) and fast (Figure 4.5b) qualities. These results have been split into two figures for readability

[17] http://blogs.msdn.com/b/visualstudioalm/archive/2015/07/20/performance-and-diagnostic-tools-in-visual-
aspx

[18] http://www.dependencywalker.com/

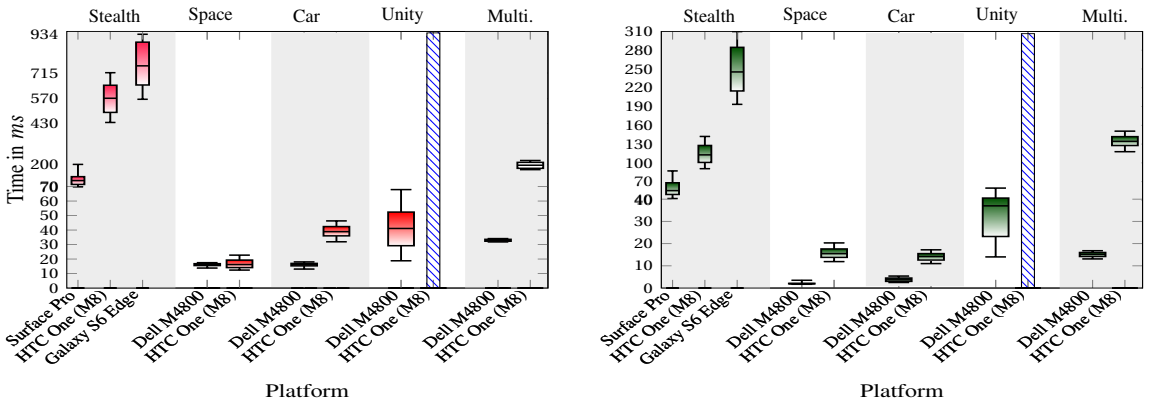
because the time needed to generate one frame for the Stealth and the Multiplayer FPS games, on the mobile devices, is one order of magnitude larger than for the other games. Moreover, Car, Unity Lab, and Multiplayer FPS have not been tested on all devices and platforms. The same remark applies for Figure 4.6, which represents the time spent by the CPU to generate one frame for the two web players on the Dell laptop.



(a) Good Quality

(b) Fast Quality

Figure 4.4: **Time required by the according processing unit to generate one frame. The box plot includes the 10th, 25th, median, 75th, and 90th percentiles of these times**

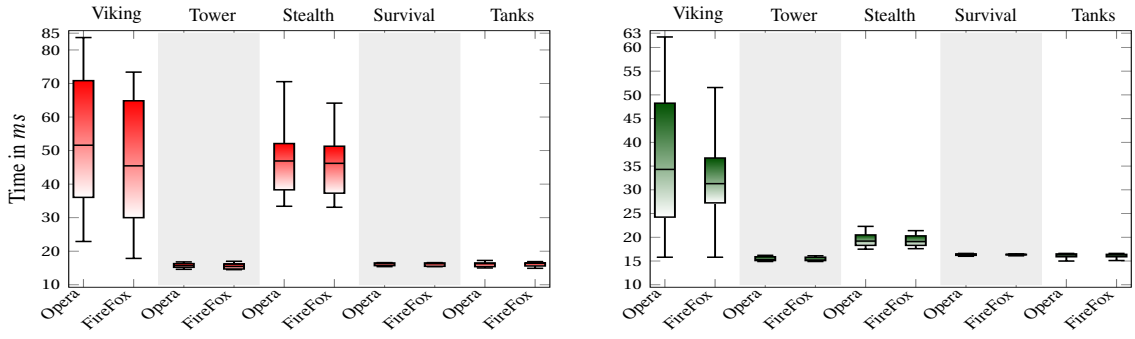


(a) Good Quality

(b) Fast Quality

Figure 4.5: **CPU-time required to generate one frame on the standalone and wearable devices**

These figures reveal a wide range of performance regarding the time needed to generate one frame. The obtained results confirm that predicting the frame generation time of a game depends on three main



(a) Good Quality

(b) Fast Quality

Figure 4.6: **Time required to generate one frame for the Web Players on the Dell M4800**

factors: the quality settings, the platform that runs the game engine, and the type of the game.

Impact of the quality settings. It is epitomized by the Stealth game, for which the time needed to generate a good quality frame is excessive on both Dell M4800 and Surface Pro (up to 100 and 200 ms, respectively), while it stays in more acceptable ranges for fast quality (up to 32 and 90 ms resp.). We observe the same impact on the other games: for instance, the time consumption per frame is reduced by a factor ranging from two to three for the Survival Shooter game. To understand the reasons behind this impact, readers are referred to Section 4.2.3 and Annex part A.2.1. A pixel resolution of 640 × 480 has typically shorter processing time than a 1920 × 1200 resolution, in particular because the additional pixels in the viewport requires more processing at the rasterization step. Another impacting parameter in high quality settings is the pixel light count, where the process of drawing geometry with lighting is repeated during the light pass step for all the pixel lights. Both texture quality and shadow also increase the processing time.

Impact of the used platform. The hardware configuration and the used OS matter. Regarding the hardware, it is obvious that: (i) using a GPU enables faster frame rendering; (ii) large memory prevents page fault exception and paging process; (iii) multi-core CPU better exploits the parallel executions. Considering the OS, each one has a different architecture with different interruptions, scheduling policies, and algorithms for page replacement in cache, central memory management, and swap. In addition, the device drivers and the used compiler may impact game performance.

For all games, the time needed to generate one frame is smaller on the Dell than on tablet, which in turn is faster than smartphones. Regarding OS, we observe a difference in the frame duration, such as Viking Village game at fast quality is three to four time faster in Linux than in Windows. For other games (e.g., Tower Bridge), the frame duration has different variations depending on the executions in Linux and Windows. We also notice that the frame generation time on the web players is (to our surprise) in the same range as when the frame is directly generated on Windows and Linux. We explain this by the screen resolution, which is automatically reduced to almost a half of the screen window. Some parameters of the quality setting are independent of the screen size, but multiple parameters directly

depend on the viewport size.

Impact of the game genre. Some games are more demanding than others with respect to scene complexity, AI, and scripting. The various game genres impact the frame duration. For the same platform and the same quality settings, two different games exhibit different profiles of resource consumption and variability of resources. In particular, we observe that the variability of the frame generation times differs among the games: for some games, all the frames take approximately the same time to be generated, while the generation of a frame can be eight times longer than another frame for some other games. Each game has a behavior signature.

4.4.2 Game Classification

Game providers are interested in determining the best option to host the game engine. From our previous results and analysis, we identify two *criteria* to characterize games: the variability of resource demand, and the relation between resource demand and quality settings, which we call *playability*. We set *thresholds* for both criteria and we define *predicates*, which can be either verified or not by each game. Finally, we classify games based on the set of validations. In the following, we first detail the predicates (criteria and thresholds), and then we explain the classification process.

Playability. It is a mix of framerate and quality settings. To evaluate a framerate for a game, we take the 90^{th} percentile of the longest frame duration.This percentile is a basis for the setting of the achieved framerate of the game. The higher is the framerate, the better is the QoE. The latter also depends on the quality setting. The higher is the quality setting, the better is the immersion, so the higher is the QoE. We combine both by setting that a given game is *playable* if the framerate is greater than 30 fps (60 fps, respectively) for good quality (fast quality, respectively) settings. That is, the 90^{th} percentile should be lower than 33 ms (16 ms, respectively) to validate the playability of a given game. This criteria depends on the platform on which the game engine runs. For example, Tower Bridge at good quality on the Dell M4800 is playable on Linux but not on Windows. When a game cannot verify the playability criteria on a device, the game engine cannot run on the device in the traditional client-server mode. Other options must be considered, either Cloud gaming platform or computation offloading.

Resource Variability. It describes the variability in frame generation time. We consider two values to differentiate high and low variabilities: the *range* and the *Inter-Quartile Range (IQR)*. The range is the difference between the highest and the lowest score, while IQR corresponds to the range of half of the scores around the median (the difference between the 75^{th} and the 25^{th} percentiles). We say that the variability is low when the range (IQR, respectively) is less than 20 ms (10 ms, respectively). For example, Survival Shooter, Tanks, and Tower Bridge games exhibit a low resource variability.

The resource variability matters because it impacts the efficiency of the implementation on a virtualized hardware. A high resource variability means that the game provider must reserve a vast amount of resources to absorb the peak, at the expense that the game engine only uses a fraction of these resources during most of the execution time. High resource variability, thus, means an inefficient

resource reservation and a lower benefits for the game providers. On the contrary, a game characterized by a low resource variability is easy to pack into a well-sized VM (or container), which enables a better consolidation and low infrastructure cost.

Based on both predicates, we classify the games as follows:

- *Client-server*, where all functions of the game engine (except session and multi-player management) run on the device. A game falls into this category if and only if it verifies the playability criteria.
- *Cloud gaming*, which means that the game as a whole runs in a VM in a data-center, while the client only runs a video decoder. A game falls into this category if both it is not playable on the device and it exhibits a low resource variability.
- *Computation offloading*, which means that the game engine is distributed among a client and a server, both of them being linked with a high-bandwidth low-latency network connection. A game falls into this category if the playability criteria is not verified and the game exhibits a high resource variability.

Table 4.4 summarizes the classification of the games based on the aforementioned devices (more details in the appendix). We observe that, for these representative games and devices, Cloud gaming is rarely the best option (only for two games on the mobile devices, and one game on Dell, and browsers). On the contrary, solutions based on computation offloading are the best option for four games in all devices except the fourth game on the Dell and browsers. This result calls for a deeper exploration of computation offloading solution, which has not been studied for gaming so far.

Table 4.4: **Best option for architecture implementation per game and device**

Games	Dell M4800	Surface Pro	HTC M8	Galaxy S6 Edge	Browsers
Viking Village	Offload	Offload	Offload	Offload	Offload
Tower Bridge	Client-server	Client-server	Client-server	Client-server	Client-server
Stealth	Offload	Offload	Offload	Offload	Offload
Survival Shooter	Client-server	Client-server	Cloud	Cloud	Client-server
Tanks	Client-server	Client-server	Client-server	Client-server	Client-server
Space Shooter	Client-server	Client-server	Client-server	Client-server	Client-server
Car	Client-server	Client-server	Cloud	Cloud	Client-server
Unity Lab	Offload	Offload	Offload	Offload	Offload
Multiplayer	Cloud	Offload	Offload	Offload	Cloud

4.5 Is Computation Offloading Possible in Game Engines?

We address now a second part of our measurement analysis with a focus on modular aspects of game engines. Our main motivation is the study of solutions based on computation offloading. In this approach,

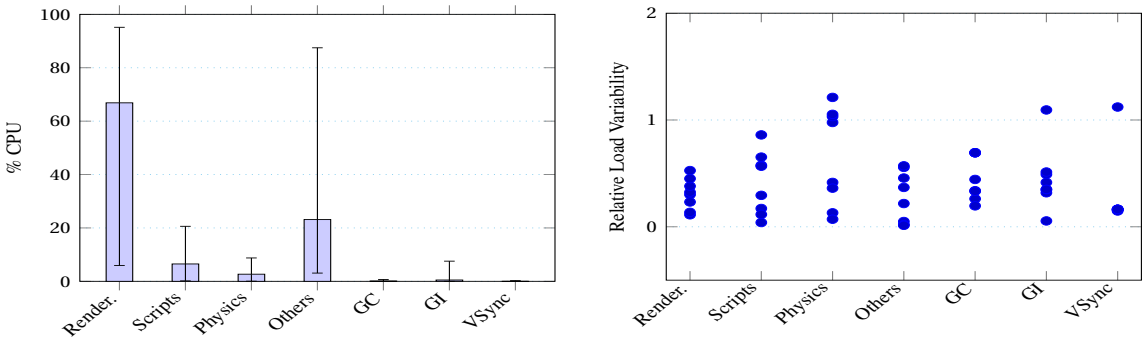
the resource requirements of each module and the interactions between the modules are two needed key information for an efficient implementation. A secondary motivation is the study of solutions based on Cloud gaming. In existing platforms [111, 174], the game engine is seen as a “black box". One of the improvements we would like to study in the future is to distribute a game engine into several physical machines in a data-center to enable parallel computation. Similarly, one needs to understand the resource requirements at the module scale as well as the interactions between the modules.

4.5.1 CPU Consumption per Module

To understand the time consumption of each module per frame, we look at the CPU usage percentage at the modular level. We gather modules into seven families: Rendering, Scripts, Physics, Garbage Collector (GC), Global Illumination (GI), Vsync, and Others. We abusively use the term "modules" hereafter to refer to a family of modules.

Figure 4.7 shows the consumption (in % of CPU) of modules. We considered for each module, the percentage of time it takes to compute in relation to the overall consumption needed to generate one frame. We aggregated all the data corresponding to the nine games, the two qualities on the different devices (except the Dell tower server) into Figure 4.7a. We plotted the three key values: average, minimum, and maximum consumption. For more details, readers may refer to Figures A.2 and A.3 in the appendix.

For every game, the frame duration is limited by one of three threads: the CPU game thread, the CPU render thread, or the GPU thread. In Figure 4.7a, we identify the rendering engine as the most consuming module. It is responsible of up to 70% of CPU usage in average. Since the rendering is done by the GPU, it means that the game performance is limited by the GPU thread. Indeed, once the CPU launches the rendering process in the GPU, it can run in parallel some other modules, including AI, physics and scripting. As soon as the process of these modules terminates, the CPU waits for the GPU process termination. For some highly-graphical games such as Viking Village, the rendering modules are responsible for 95% of the CPU consumption on the Dell Win7 platform at high-quality.



(a) Module Consumption (min, avg, max)

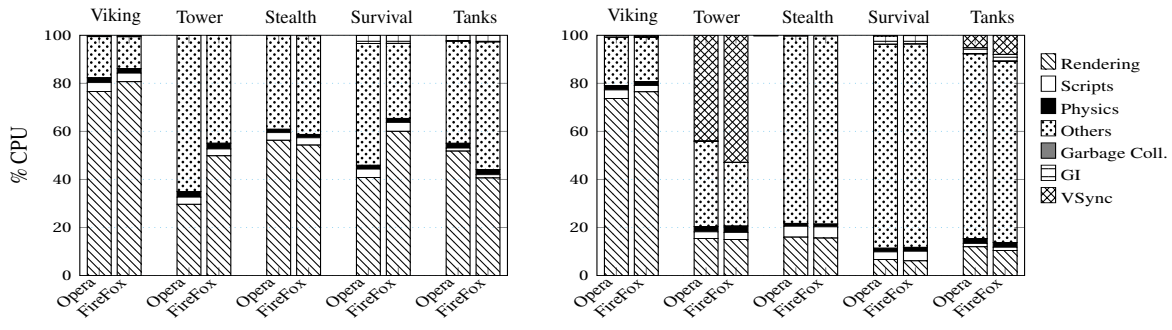
(b) Relative load variability: $\frac{r_{pq}-R}{\mu}$

Figure 4.7: **Module resource requirement and relative load variability**

We are also interested in understanding the differences in the module consumption *per platform and game quality*. For a given platform and a given user quality setting, we compute the average time consumption for all frames and all games to obtain a global average platform-dependent quality-dependent module requirement. We thus get eight measures of the average module requirement (four platforms and two game qualities), noted r_{pq} for each platform p and each quality q. We compute the average requirement, noted R for the eight measures. Then, we compute, for one given platform-quality pq, the difference to the average measure, $r_{pq} - R$. A platform-quality pq with a high difference means that, for this given module, the requirement significantly differs from the other platform-quality, and thus it would justify the game provider to pay a specific attention. Our idea is also to check if some modules exhibit much wider different behaviors than others. To enable comparison across modules, we thus compute a *relative* measure of the difference by computing the ratio of this difference $r_{pq} - R$ to the standard deviation μ. We call this measure the *relative load variability*. The higher is the relative load variability of a platform-quality pq, the more different is the platform-quality pq from the other platforms.

We represent, for each module, the eight relative load variability measures in Figure 4.7b. We have two observations. First, the resource requirement is sensitive to platform-quality. Some platform-quality pq have a difference to the average that is nearly 1.5 larger than the standard deviation. It calls for paying attention to platform-quality when implementing a computation offloading because these platform-quality exhibit specific and unusual resource consumption. Second, all modules have similar relative load variability. This is counter-intuitive since we expected that rendering modules would be more sensitive to platform-quality than other modules.

In Figure 4.8, we represent the consumption per module for the web players. The CPU spends a lot of time computing other type of modules, like scheduling the tasks over the different layers. The rendering portion of time is lower than for the case of standalone platforms, which is due to the default resolution used in the web players. The script and physics take more portion of CPU than in the standalone cases.



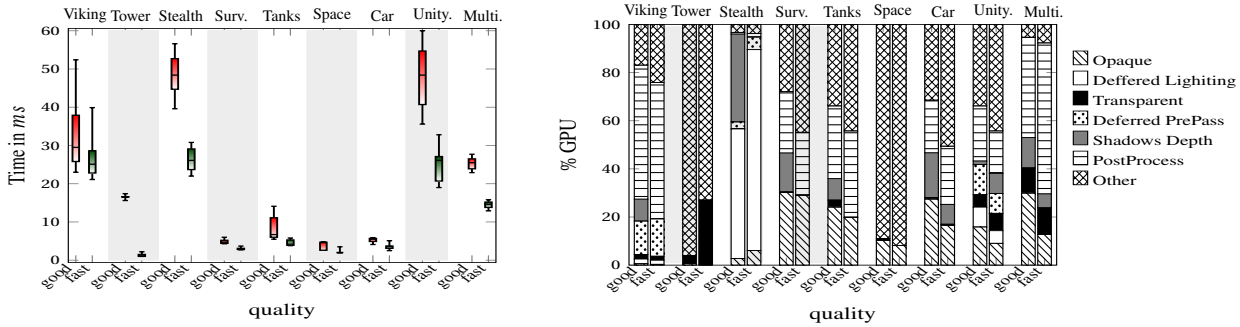
(a) Good Quality

(b) Fast Quality

Figure 4.8: **CPU consumption per modules on Dell M4800 for Web Players**

4.5.2 GPU Consumption per Module

We concluded from Figure 4.7 that the rendering modules are the most consuming modules regarding CPU use. One of our previous explanations is that CPU is bounded by the performance of GPU to render the scene. In the following, we look at the GPU consumption to validate this statement. We plot in Figure 4.9a the time spent by the GPU to generate one frame for each of the games on the Dell laptop. In Figure 4.9b, we show the percentage of time the GPU spends in each subfamily of the rendering module. We derive from Figure 4.9a that the time taken by the GPU is approximately the same as the CPU in



(a) Consumption per frame

(b) Consumption per module

Figure 4.9: **GPU consumption per frame and per module on Dell M4800 for Windows 7**

Figures 4.7a. It confirms our intuition that, since the CPU finishes processing tasks faster than the GPU, the CPU waits for the GPU to finish the frame generation. As we will see later, the “waiting mode" is considered as a module to schedule and synchronize the two processors. Moreover, in Figure 4.9b, the consumption per rendering activity is different for each game. Indeed, the nine game worlds are different. We also observe that the GPU is not entirely used for rendering processing. Typically a game like Tower Bridge consumes the entirety of available GPU for other types of module.

4.5.3 Calls between Modules

We pay now attention to the modular architecture and the interactions between these modules. This study is key for the implementation of computation offloading, and it is also useful for new implementations of Cloud gaming systems.

We depict in Figures 4.10 and 4.11 the main calls inside each family of modules. Specifically, Figure 4.10 (4.11a, 4.11b, 4.11c, 4.11d, and 4.12, respectively) corresponds to the Physics module family (Audio, AI, Animator, Script, and Rendering module families, respectively). In each graph, the vertices represent each module in the family, while the oriented edges are a combination of two parameters: the first one is the number of time the source vertex calls the destination vertex (the calls frequency), the second value is the execution time of the destination vertex. These flowcharts correspond to the case of Survival Shooter game on the HTC One (M8) for the good quality encoding.

Each module family can be offloaded as a separate service. Indeed, we observe in the figures that these families exhibit few interactions and data sharing *each other*, which is expected since these

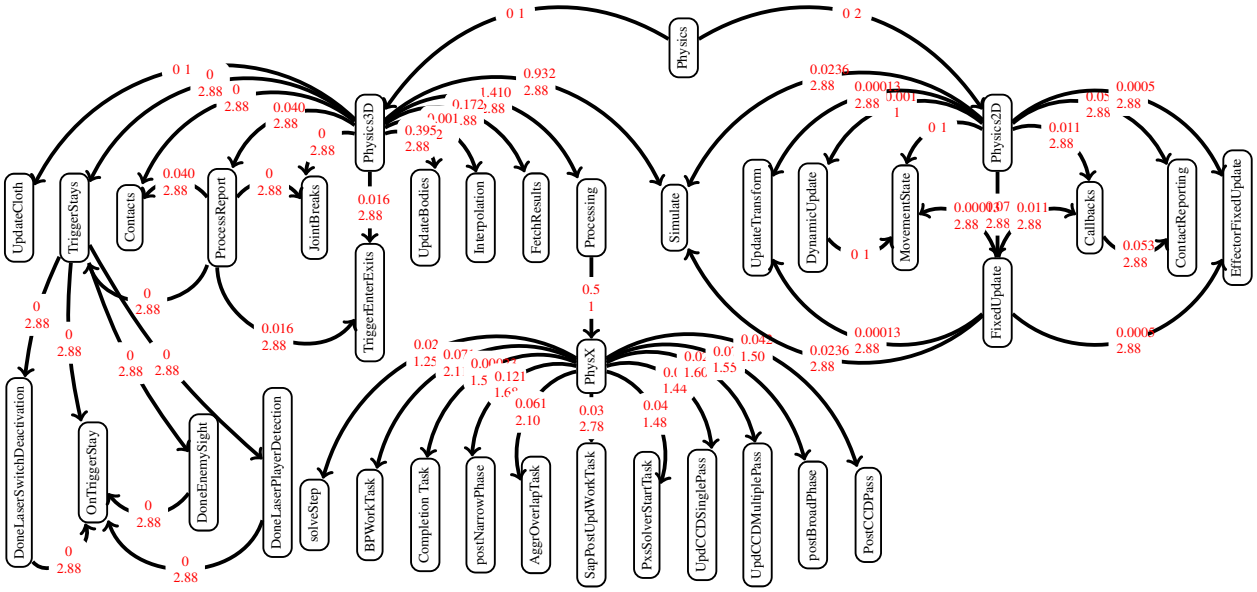


Figure 4.10: **Internal calls inside the Physics module**

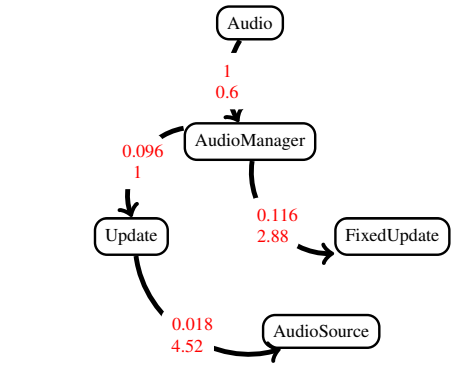
families of modules are often designed and implemented by different teams. The main coordination and synchronization task is done through the modules *Update* and *Fixed Update*, which are shared between all the families and represent the main game thread. Since the frequency calls between families are low, the offloading per module families makes sense. Inside each module family, here are our main observations:

- The Physics family (see Figure 4.10) can be spread into three sub-families: *Physics3D*, *Physics2D*, and *PhysX*. PhysX interacts only with Physics3D over the *Processing* class, and Physics3D shares with Physics2D only one class (the *Simulate* class). It would thus be relatively simple to distribute the computation of these three sub-families over distinct computers.
- The Scripting family (see Figure 4.11d) is the only module that interacts directly with the Physics engine. It would thus make sense to run this family of modules on the same computer as the one that hosts the Physics modules.
- The Audio family (see Figure 4.11a) has no interaction with the other main modules and only deals with the game thread. This module can thus be offloaded as an API to a remote machine. The communication between the client and the remote machine can be done either by RPC or by streaming the audio data.
- The AI family can be computationally intensive. It is difficult to simulate the game on constrained-resource devices without reducing the complexity of the AI. Since this family is also mostly independent, the module family can be offloaded.

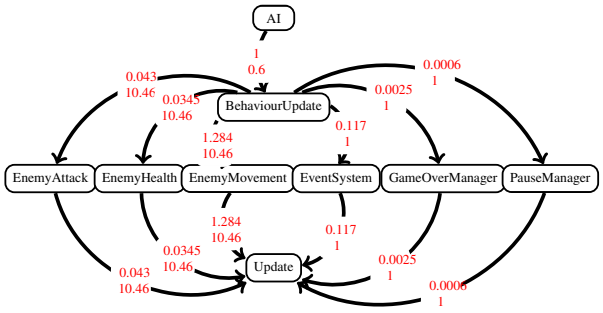
Finally, we address in Figure 4.12 the Rendering modules. We revealed in our previous results that

this family is the most resource consuming, and thus is a candidate to migrate from the lightweight client devices to nearby servers in a computation offloading solutions (or from one standard server running the game engine to a specialized GPU-enabled server in the same cluster of servers for the case of a Cloud gaming solution).

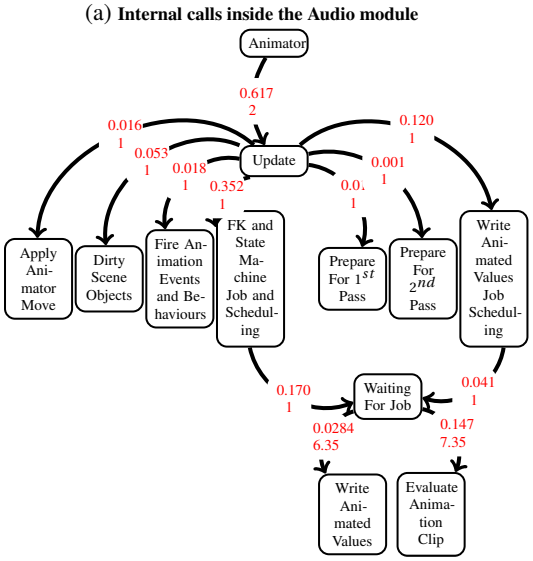
The offloading mechanism can be instruction-based or image-based. In instruction-based systems, the client renders the graphics by itself using the commands received from the server. This system consumes less bandwidth as only graphics drawing commands transit over the network. Image-based system streams the rendered game as a real-time video. The clients are platform- and implementation-independent, and demand fewer resources. However, it is harder to distribute this module. Indeed, we identify some “sub-families" within the Rendering family that are rarely independent. We observe in Figure 4.12 that the calls come from multiple other modules, which are not necessarily in their own sub-family. These inter-calls between modules, from different sub-families, make the code offloading harder. Indeed, the calls frequency is high, which implies intensive communication between the different computers and ultra-low-delay data mirroring in the case of offloading. Moreover, some sub-families are more called than the others. For example, *SharedSet-Pass*, *RenderTexture*, and *MeshVBOModule* are called to generate the frames. Finally, the *WaitingForJobs* module is used to synchronize the modules that have different running time. The time spent into the WaitingForJob module is a waste of time and CPU consumption.



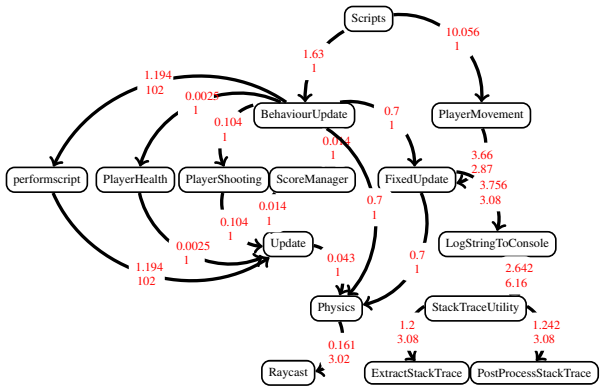
(a) **Internal calls inside the Audio module**



(b) **Internal calls inside the AI module**



(c) **Internal calls inside the Animator module**



(d) **Internal calls inside the Scripting module**

Figure 4.11: **Internal flow concerning the main modules: Audio, AI, Animations, and Scripting**

4.6 Game Performance when Assisted by Server.

In Section 4.4, we tested the different games in the traditional client-server architecture, from which we derived a game classification for the different platforms. In Section 4.5, we studied the possibility to offload a game engine and concluded with promising solution regarding offloading the game engine modules. Now, we present the performance of the games when we offload a part and the whole game engine to a server as in the two architectures (Cloud gaming and computation offloading). We present the needed CPU-time to generate one game frame on the client device in both architectures.

For a fair comparison between the games, we offload the same game objects namely NPC, Player Character (PC), environment, and lights. Table 4.5 summarizes the game module distribution. Table A.4 presents the used devices for each architecture.

Figure 4.13 depicts the time needed to generate one frame on the three architectures (i.e., client-server, Cloud gaming, and computation offloading) for each of the used games for the good (Figure 4.13a) and

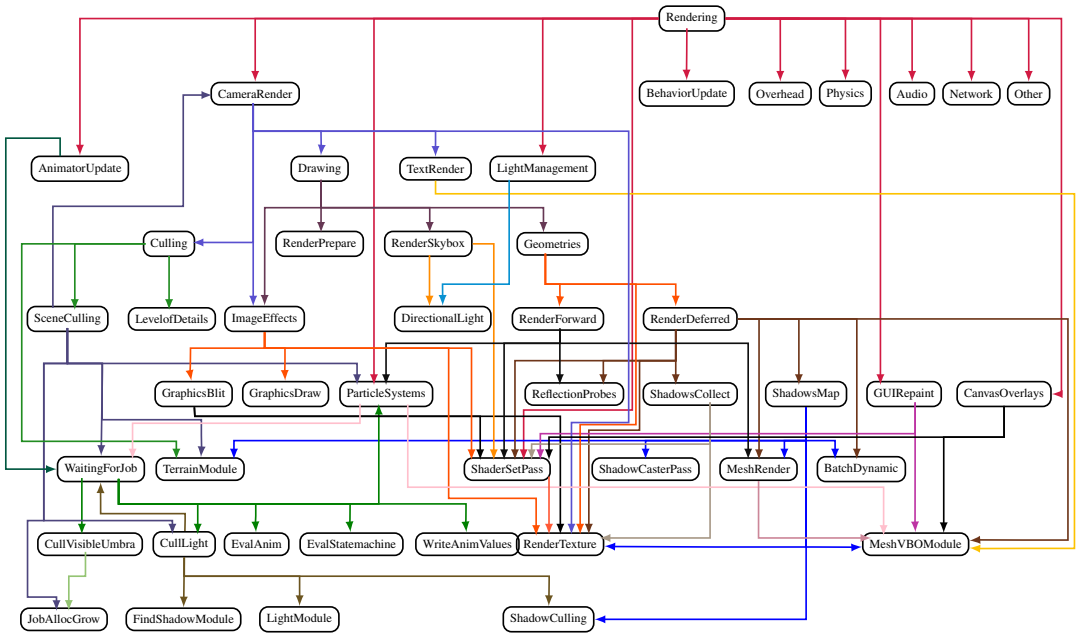


Figure 4.12: **Internal calls inside the Rendering module**

Table 4.5: **Location of the game objects in case of computation offloading**

Element	Module	Client	Server
Player character	Rendering	☒	☑
	Audio	☒	☑
	Animations	☒	☑
	Scripting	☒	☑
	Physics	☒	☑
	Inputs	☑	☒
Non-Player character	Rendering	☒	☑
	Audio	☒	☑
	Animations	☒	☑
	AI	☒	☑
	Physics	☒	☑
Particle Effects	Rendering	☑	☒
	Animation	☑	☒
	Physics	☑	☒
Arena	Rendering	☒	☑
	Physics	☒	☑
Sound	Audio	☑	☒
	Scripting	☑	☒
Sun Light	Rendering	☑	☒
Arena Light	Rendering	☒	☑
Main Camera	Rendering	☑	☒
Weapon Camera	Rendering	☒	☑
Game Manager	Scripting	☑	☒

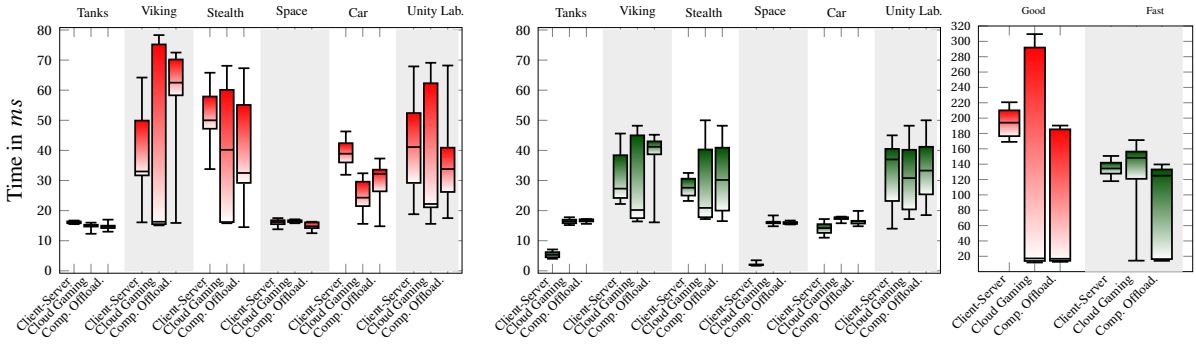
fast qualities (Figure 4.13b). We use box-plots to present the results since we focus on the variability of CPU consumption per frame. Indeed, the *consolidation* of resources in a data-center is easier when the consumption of processing resources is accurately predicted. The more stable are the CPU and GPU consumption, the more games can concurrently run in a cluster.

We distinguish three categories of game. Some games are ideal for client-server architecture and/or consolidation because all frames take approximately the same CPU and GPU time to be generated. The

10^{th} and 90^{th} percentiles are so close that they nearly overlap. It is notably the case of Tanks and Space Shooter. A game offering a small variability features scenes that are not complex and are thus easily rendered by the GPU. For the good quality, a frame is generated in less then 33 ms and in less then 16 ms for the fast quality. These games are device-friendly.

Some other games exhibit a high variability, notably Viking Village, Stealth, and Unity Lab. These are the worst cases for Cloud provider, which has to reserve resources to accommodate the peaks (more than 75 ms CPU for Viking Village, 65 ms for Stealth, and 67 ms for Unity Lab), although the median frame requires almost quarter the time (16 ms here). The reserved resources are wasted. These games in general are the most consuming for the CPU and GPU resources due to the scene complexity. Since these games are not desirable for Cloud gaming architectures, and cannot be played locally on the wearable devices, then these games should be offloaded.

Finally, the games that are the less demanding and have less variability, typically Car in our set of games, are good candidates for Cloud gaming systems.



(a) Good Quality (b) Fast Quality (c) Multiplayer Game

Figure 4.13: **CPU-time required to generate one game frame on the three architectures**

4.7 Discussion and Main Findings

To sum up our findings in this chapter: Firstly, the GPU is the main source of performance limitation in the different cases. The CPU game and CPU render threads are frequently blocked waiting for the results of the GPU due to a *synchronization* problem. To improve GPU performance, an idea would be to free the latter from some jobs. For instance, recent GPU cards improves the game performance by offloading the PhysX from GPU to CPU, especially for powerful CPU. Unfortunately, this solution works only for games using PhysX as physics engines (like Batman [27], Assassin's Creed [202], and some others[19]) but it does not work for other engines like Havok, ODE, or I-Collide. We also identified that both CPU and GPU consumptions have a high correlation, which is useful for the Cloud gaming

[19] `http://www.geforce.com/hardware/technology/physx/games`

providers to estimate from the CPU demand the amount of GPU resources that need to be reserved, as well as the game variability.

Second, the frame rate of device-friendly games is generally higher than 60 fps. This is a waste of resources, especially in terms of energy. Here, we envision a solution where we first calculate the number of generated and saved frames in the frame buffer, by monitoring the synchronization, then the GPU breaks the game main loop when the number of 60 fps is reached. The idle GPU saves a signfiicant part of the energy consumption. The provider can use the idle GPU to serve more games using a preemptive scheduling. NVIDIA proposed the mechanism *G-Synch*, which improves the *V-Sync* mechanism, where the monitor refresh is conducted by the GPU frequency. It finds trade-off between the off- and on-mode of V-Sync. Indeed, when V-Sync is deactivated, the GPU sends the frames following its own pace. At each screen refresh cycle, multiple frames are displayed because the frame rate is maximal causing the phenomenon of *tearing*. Now, when the V-Sync is activated, the GPU follows the screen tempon so tearing no longer occurs. But, when the time to render a frame is longer than a refresh cycle, the phenomenon of *stutter* and *display delay (lag)* occurs because the monitor has to display again the last frame.

Third, some modules related to rendering are mostly in waiting mode, meaning that the CPU consumption associated with these modules is not significant. These waiting times are not necessarily a waste of resources when only one game runs on a computer, since the rendering pipeline is at a given step and no further actions can be taken. However, in the context of Cloud computing, these waiting times represent opportunities to free some resources and to better exploit the processing units.

Fourth, a non-graphic component of the game engine can represent a significant part of the CPU consumption, typically in games where the AI, the physics, and the scripts are complex. A motivation behind code offloading is to study the gains when the display device embeds dedicated GPU resources, and powerful servers are available nearby. This configuration matches the new generation of game centers with VR headsets.

Finally, the game classification of the games into the three architectures matches our predictions based on the criteria: *playability*, and *resource variability*.

4.8 Conclusion

This chapter has dealt with the topic of game engines. We presented a general architecture and conducted a set of experiments on three architectures: client-server, Cloud gaming, and computation offloading. We used nine representative games, including FPS, TPS, Racing, and MMOG. We adapted these games to different platforms and tested their performance on each platform for two encoding qualities. Based on our results, we classified the games into the three architectures depending on two parameters: playability and resource variability. We also identified that the quality settings, the used platforms, and the game genre exhibit varying behaviors. Next, we provided a detailed view of the game engine by representing the internal flow that constitutes each module, which is a basis for solutions exploring

computation offloading. Finally, we validated our finding regarding the game classification according to the aforementioned criteria and the feasibility of game engine offloading. In our experiments, we used *static offloading* of the sub-families of modules. Our work opens new exciting perspectives and research directions to improve gaming experience. Indeed, we may mention particularly two directions to follows, one based on computation offloading and another one on Cloud gaming consolidation. Regarding computation offloading, it is interesting to explore not only partitioning algorithms to find the best execution location of each module sub-family, but also use systems for polygon rendering to decompose an object, or an entire image to offload some part aiming at improving the rendering process between the client and the server. About game consolidation, it will be useful to define a regression model for the resource consumption to maximize the consolidation and minimize the risks, and find somehow to serve different gamers in Single Online Shared Game Instance (SOSGI) (i.e., throw only one instance of game engine).

CHAPTER 5

TOWARD A MOBILE GAMING BASED-COMPUTATION OFFLOADING

5.1 Introduction

In light of conclusions made through chapter 4; video games are complex, intensive, and real-time applications that need powerful devices to run smoothly, in order to provide a gamer an illusion of inside an animated world; the rendering engine should perform all the rendering activities in real-time. For a gamer, a good QoE is obtained when the game engine displays frames at a higher rate, more than 30 fps [54]. Thus, the rendering engine has at most 33 ms to generate one frame. Usually, much less time is available, since the bandwidth is also consumed by the other engines such physics and scripting. Inside the umbrella of conclusions, we also demonstrated that computation offloading is feasible for game engines. Indeed, we extracted the different call flows inside the game engine and checked the possibility of splitting modules. We observed that module families are mostly, independents. Therefore, it is possible to offload these modules entirely or partially with respect to the network latency. The classification of games into three architectures namely, client-server, Cloud gaming, and computation offloading revealed that computation offloading architecture *should be* the best option in most cases. This observation was existing and a good motivation to go farther and provide a static solution for computation offloading of game engines.

Through this chapter, we unveil an enhancement of our game engine offloading proposal by making *dynamic*, the offloading decision. We propose a heuristic, which according to the network latency, resource consumption, and code dependency, selects clusters of modules that should be offloaded to improve the performance. We survey state-of-the-art computation offloading-oriented mobile gaming. Then, we introduce some concepts and keywords that we use through the chapter. After what, we describe our methodology and contribution. We use a testbed composed of a smartphone and a server to evaluate and compare the performance of our solution with the classical one.

5.2 Related Work

Computation offloading is gaining ground with the emergence of MEC, which locates servers at the network edge allowing to drastically reduce the end-to-end latency. Several works have addressed the problem of computation offloading in the context of mobile Cloud. We cite hereafter some of these works (reported in the previous chapters) oriented for games with some technical implementation details. In *MAUI* [59], the authors presented a dynamic offloading framework operating at the method (i.e. application component) granularity. The games are represented with graphs and translated into a linear programming formulations, with the objective to save energy, subject to the total execution time. MAUI requires annotating methods as “remote" or “local". The communication between the mobile device and the Cloud is done through RPC. The design of MAUI supports only games written in C#. Similarly, *ThinkAir* [139] introduced a mobile Cloud computing framework, with dynamic offloading. The proposed framework clones the smartphone platform in VM. It offers a library and a compiler to ease the adaptation of games. A code generator creates wrappers and utility functions. A customized Native Development Kit (NDK) is used to convert the ARM-based instructions of remote methods into x86 instructions. ThinkAir uses *Java reflection* to offload methods based on past invocations. ThinkAir defines four objective functions that combine execution time, energy, and money cost. In [323, 324], the authors propose *DPartner* framework, an automatic partitioning system that rewrites Java bytecode of monolithic application into a distributed application. The framework operates in three steps; first, it classifies Java bytecode classes into anchored or movable based on Java lexicon. Then, the framework clusters classes regarding the call frequency into different groups representing the game modules. Finally, the framework rewrites the clusters bytecode and packages them into OSGi bundles. These bundles are classified into anchored or movable modules according to classes. The framework defines a proxy, which rewrites the classes to create new interfaces, and duplicates classes on both sides (client and server). The framework offloads all the bundles that improve the performance and reduce the energy consumption. Another framework proposing to offload GPU computation to remote servers has been introduced in *Kahawai* [60]. It uses a collaborative rendering, which combines both server GPU and mobile device GPU outputs to render frames. Kahawai uses two techniques for collaborative rendering; the delta encoding and client-side I-frame rendering. In delta encoding, the mobile device renders frames with low quality encoding, while in the server, the same frames are rendered with high quality. The per-frame difference is streamed to the mobile device to transform frames into high quality frames. In client-side I-frame, both the mobile device and server are rendering frames at high quality encoding, however, the mobile device generates the frames at a low rate compared with the server. The server compresses the frames into a video, replaces the I-frames with empty place-holders, and streams the remaining P-frames to the mobile device, which fills in the missing I-frames and renders the video. Di et al. [112] have proposed *Dust*, a real-time code offloading system for device-to-device. Dust uses a network evaluator component to find the stable linked offloadees, and a task scheduler component, which takes a decision regarding each task of a game. The tasks are annotated by programmers as “@offloadable".

Even these frameworks provide in general good performance, these framework are obsolete for the use case of 3D FPS games, which are very high resource consuming and real-time interacting. Indeed, in all these frameworks, the authors have considered only strategic 2D games like Sudoku, Chess, N-Queens, and Gomuko, which renders one frame for each player movement. To this purpose, we propose an offloading 3D FPS game system using Unity 3D game engine. We split the game scene into different Game Objects (GOs) including, NPC, PC, environment, and particles. To select the GOs to offload, we use a heuristic relaying on three main criteria, namely, resource consumption, code dependency, and network latency as detailed later. Furthermore, we introduce a network manager component, which orchestrates the offloading mechanism. This solution offers a promising performance. It is easy to configure, as the GOs to offload are added to the server through a network manager, and scalable as all games could be modified easily to fill in this architecture. Lastly, the proposed solution is not bandwidth consuming, as only command packets are exchanged over the network.

5.3 Game Engines Background

To better understand our contribution in this chapter, we introduce/remind in the following some concepts and keywords that we will use through the chapter.

5.3.1 Interactivity and Framerate

The *interaction delay*, defined as *the elapsed time between a user action is captured by a HID, and the moment that the result of this action appears on the screen*, is central in gaming. Studies [55, 160, 229] have shown that the acceptable delay depends on the game *genre* and varies from 100 to 200 ms and even up to 500 ms for RPGs and MMOGs. FPS games require low delays (less than 100 ms), since the gamer is immersed in the scene, and a high interaction delay will degrade the QoE [121]. Regarding this constraint, the management of FPS games has received scientific efforts [12, 165]; therefore the interaction delay will be considered as a metric in this chapter.

The *framerate* is another key criteria to ensure high QoE for the gamers. Indeed, the latter are immersed in an animated world when the game engine generates a high number of fps. Less than 30 fps is seen as non-tolerable by players [54]. For the interactive multimedia, the High Frame Rate (HFR) combined with the High Dynamic Range (HDR) technology can deliver up to 240 fps [153]. The *rendering pipeline* [225] represents the engine responsible for generating frames; every $x\,ms$ the graphic pipeline displays one frame, with x ranging from 33 to 10 ms. The framerate will be also considered as a metric in this chapter.

5.3.2 Main Modules

A game engine is a combination of different modules depending on the game *genre*. Messaoudi et al. [192] have identified some modules that are common to most game engines, and classified them into

different families. Some of these module families are written by game developers that include: (i) the *AI*, which emulates an artificial and intelligent behaviour of the NPC to learn, to interact, to fight, and to survive; (ii) the *scripts* represent the game scenario. Game developers detail, in a scripting language, the control flow of the game, from the instant wherein the gamer command is captured by HID until displaying a frame on the screen. (iii) *Animations* are used to make objects, dynamic in the game. They emulate movement or reshape objects.

Some other families of modules are leveraged as a third-party SDK and middleware accessible through APIs. These families of module represent an abstraction layer common to all games created within a given framework, aiming at preventing the game developers from spending time in low-level programming. These modules include the following: (i) *physics*, which simulates the physics laws to make the game as realistic as possible. Physics uses collisions and rigid body dynamics[1]. Without physics module, objects would interpenetrate, leading to block interactions with the virtual world. (ii) *Multimedia rendering* modules are responsible for generating the graphical and audio elements of the game. Rendering is a resource-consuming module in game engines, since the 3D-scene undergoes several transformations through the rendering pipeline before getting displayed on the screen [190]. (iii) *Inputs* convert the physical commands applied by the gamer on his HID (including gamepad, joystick or keyboard) into logical game functions, and forward them to the engine system. Finally, (iv) *networking* modules define a set of routines and protocols that enable interactions with a remote server to share a game instance between multiple players.

5.3.3 Scene Representation

A real-world scene is a projection of dynamic foreground (the *dynamic GOs*) on a layout of a static background or *static GOs*. The static background layout is crucial in video games, as it brings the player inside an immersive world. The game's world populates different types of GOs, through which the gamer explores the virtual world. The game world as a whole presents perceptual stimuli to the player, which experiences a degree of presence over the objects of this world that he can manipulate. These objects include: (i) a *PC*, which is a fictional character, controlled by the gamer. Generally, these characters are based on real persons, such as sportive and historical persons. FPS games use black characters without any characteristic. (ii) A *NPC* is controlled by the computer through an AI and triggered by specific actions. A NPC may define an *enemy*, a *partner* or a *support* character, depending on whether the NPC opposes the PC in duels, helps the PC in its adventure or assists the storyline of the game. (iii) *Environment* represents the virtual static and realistic area where the game takes place. (iv) *Lights* are a key step to produce a realistic scene. The light sources are simple objects, defined in the world space, which are a combination of color, intensity, direction, focus, and position. (v) *Particles* are amorphous objects such as smoke clouds and sparks. They are animated in a rich variety of ways that vary in position, orientation, and size from frame to another. (vi) *Sound sources* are in charge of reproducing

[1] `https://gafferongames.com/post/physics_in_3d/`

what the player would like to hear such as a car engine sound or a background music. (vii) *Camera* is a GO that displays what it currently *seen* on the screen. The camera can move and rotate around, hence the displayed view moves and rotates accordingly. The area seen by the camera defines a truncated pyramid known as a *frustum*.

5.4 Methodology

We describe now the game, platform, and encoding qualities that have been used in our experiment, as well as the methodology undertaken to offload modules to a remote server.

5.4.1 Game

We modified the multiplayer FPS game[2] to make player characters fighting together against a NPC inside an arena. The player character is a robot with blasters flying inside the arena. The NPC is a humanoid avatar triggered by the player characters when they are near to its position. The game scene is depicted in Figure 5.1. We summarize the main characteristics of this game in Table 5.1.

Figure 5.1: **Game screenshot**

Table 5.1: **Game characteristics**

# of players	Dimension	Type	Rendering	Physics	Scripts
Multiplayer	3D	FPS	++++	+++	+++

5.4.2 platform

To evaluate the performance of the proposed solution, we installed Unity 3D engine v5.4 on top of a Dell PC tower. The installed engine is used to compile the tested game and generate two different instances; the first one runs on the server (Dell PC tower), while the second one runs on the smartphone HTC One M8. The configuration of these devices is given in Table 5.2.

[2]https://www.youtube.com/watch?vŪK57qdq_lak

Table 5.2: **Platforms characteristics**

Platform	CPU	GPU	RAM	OS
HTC one (M8)	Quad-Core 801 Snapdragon, 2.3GHz	Adreno 330	2GB	Android 4.4.2
Dell PC tower	Intel Core i7, 3.4 GHz	3x NVIDIA GeForce GTX 780 Ti, 3GB	16GB	Windows 8.1 Pro

5.4.3 Quality Encoding

We generated 10,000 frames for two encoding qualities; a *good* and *fast* quality. The good quality is encoded with high parameter settings, which generate a reasonable framerate, i.e., around 30 fps. The fast quality, configured with reduced requirements, produces inferior visualization results, hence obtain a maximum framerate. Unity 3D achieves these two qualities through different parameters as described in [190].

In light of what is stated in the background section, the following paragraph tackles the questions: *how we can improve the performance of a game through modules offloading?* or *what is the optimal location (on the mobile device or on the server) of each module of the game engine?* To answer efficiently this question, we introduced the following criteria:

1. ***Resource consumption.*** Usually, the gameplay is concentrated within dynamic objects, which are high resource consuming. Rendering these GOs is complex in video games [190]. Each object in the scene is approximated by triangle meshes. The more triangles are used to approximate an object, the better is the approximation, but more is the processing.
2. ***Code Dependency.*** Games depend on hardware (e.g., sensors) and software known as *libraries* and *SDKs*, but also interact with players via *UIs*, which manage the HIDs. According to this, we distinguish three classes of *non-transferable modules*; modules involving UIs [59, 217]; modules interacting device sensors [216]; and modules depending on local APIs [94, 220].
3. ***Bandwidth consumption and Network latency.*** Some GOs, if they are offloaded to the server, need high network communications with the mobile device, which increases the bandwidth consumption and the interaction delay. Particles are an example, they are 2D images generated and animated in large number. Several modules are interacting together to make their behaviour, which leads to high communication between modules of the game engine.

5.5 Proposed Framework

Figure 5.2 presents a global view of the proposed architecture. At the beginning, a connection is established between the mobile device and the server via a network manager. This latter is a set of scripts responsible for remotely instantiating GOs, and orchestrating the offloading process.

When the connection is established, a *game manager* script is executed on the mobile device. It starts the execution of the local GOs, while it requests the network manager to start computation on the server. Therefore, remote GOs are rendered on the server at default coordinates. Start playing the game, input modules capture the gamer inputs and send commands to the server. On the server side, both the GOs and the outputs of the involved modules (modules used by each GO to compute its behaviour) are updated with the gamer commands. Thus the network manager captures these results and injects them on modules, located in the mobile device, interacting with the remote GOs. At the end, a frame composed of the local and the remote GOs is rendered on the mobile device. This process is repeated for each frame until a disconnection of the gamer. In this case, the GOs are destroyed on both client and server.

The network manager uses both *RPC* and *GigE Vision Stream Protocol (GVSP)* [103] carried over User Datagram Protocol (UDP) for communication between the mobile device and the remote server, with a multi-channel design supporting a variety of levels of QoS, and a flexible network topology supporting peer-to-peer and client-server architectures. RPCs are used to update some module entries and variables such as (N)PC *health level* or *weapon state*. GVSP is used to stream *OpenGL ES* commands from the remote server to the mobile device, then the latter will prefetch these commands and inject them inside the rendering pipeline to draw a frame on the mobile device.

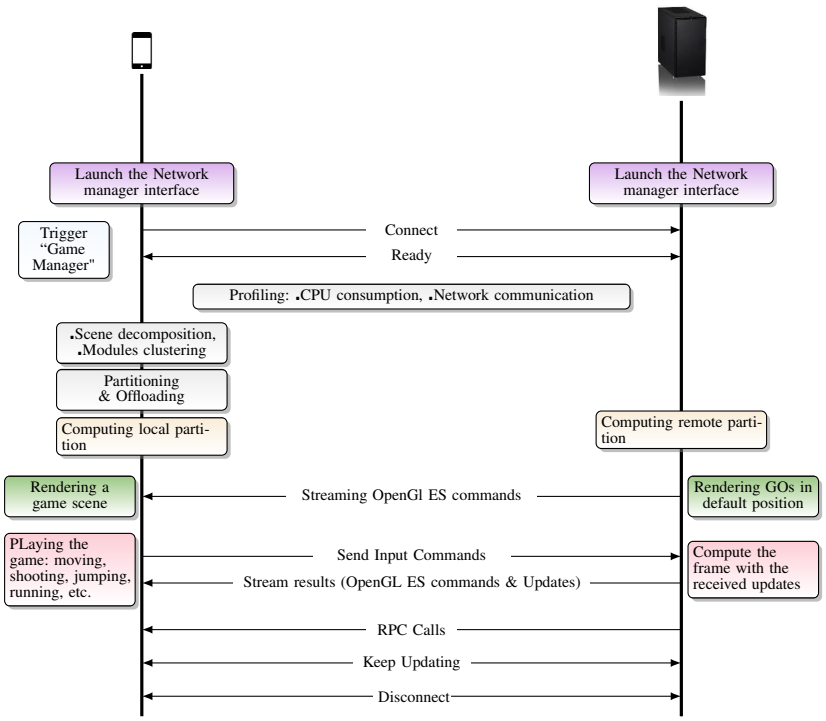


Figure 5.2: **Global overview of the architecture**

5.5.1 Proposed Heuristic

Each GO involves a number of modules to compute its behaviour and to draw its shape. These modules may differ in number and family between the GOs. We propose to enclose for each GO, the requested modules inside clusters. Our heuristic, computed by the network manager, dissects the possibility to offload or not a cluster, with respect to the code dependency constraint. This solution is a cluster decision-making; that is, it focuses on each GO independently from the others, since we need to decide rapidly to offload a cluster or not, to avoid any additional delay induced by more sophisticated algorithms like ILP or graph resolution. The proposed algorithm accepts as input a cluster, j (the set of modules requested by the object GO_j) and returns a binary decision x_j (i.e., to offload or not the cluster j). The concept of this algorithm is simple; if either the network latency or the time to send/receive data is higher than the local execution time of a cluster, then offloading the cluster will not improve the performance, thus $x_j = 0$. Otherwise, if both latency and time to exchange data are less than the cluster execution time, the algorithm checks if a gain is achieved when offloading the cluster. The *offloading gain* is the difference between the local cost and the offload cost, this latter includes the communication cost and the remote execution. It is given by:

$$TG_{GO_j} = \left(\frac{s-1}{s}\right) \times \sum_{i=1}^{k_j} T_i - \left(\frac{d_i}{UL} + \frac{r_i}{DL} + RTT\right) \tag{5.1}$$

where s is the speed ratio between the mobile device and the remote server, T_i is the execution time of the module i that belongs to cluster j, k_j is the number of modules enclosing the cluster j, and d_i and r_i represent, respectively, the data to send (receive, respectively) on the uplink, UL (downlink DL, respectively) bandwidth.
If offloading a cluster will achieve a gain, then it will be offloaded (i.e., $x_i = 1$).

Algorithm 1 Offloading Decision

Inputs: $GO_j : \{T_i, i = 1, 2, ..., k_i\}$
Outputs: $Decision\, x_j$
if $((RTT < \sum_{i=1}^{k_j} T_i)$ & $(d_i/B_s + r_i/B_r < \sum_{i=1}^{k_j} T_i))$ **then**
    if $(TG_{GO_j} > 0)$ **then**
        $x_j = 1$
    else
        $x_j = 0$
else
    $x_j = 0$
return x_j

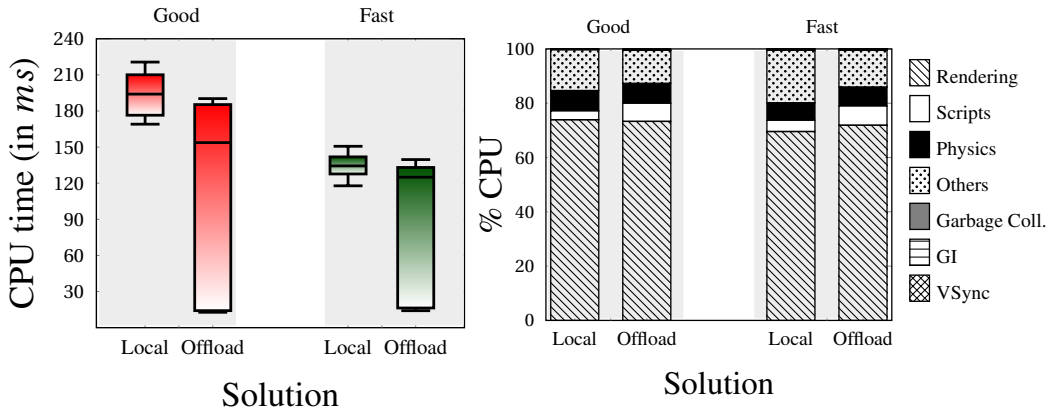
5.6 Performance

Now, we present the results of our measurement campaign regarding the CPU consumption and network communication needed to generate one game frame.

5.6.1 CPU Consumption

This section discusses the CPU consumption per frame and per module, for local and remote execution, under the two encoding qualities. Figure 5.3a presents the time (in ms) needed to generate one frame. We used box-plots as we want to focus on the *variability* of the CPU consumption per frame. The aim is to quantify the *stability* of our framework. The box plot includes the 10^{th}, 25^{th}, median, 75^{th}, and 90^{th} percentiles of these times. On the other hand, Figure 5.3b shows (in %) the time spent by the aforementioned modules to contribute to the frame generation. We observe two things:

- *Performance improvement*. As seen in Figure 5.3a, our framework improves the performance by up to 21%. Indeed, more than 50% of frames are generated in less than 154 ms (125 ms, respectively) for the good (fast, respectively) quality. However, the framework is not enough stable as the *IRQ*[3] and the *range*[4] are high values (143.97 ms and 151.52 ms, respectively).
- *Rendering consumption*. As Figure 5.3b is showing, rendering is the main consuming module. Indeed, this module is responsible for up to 70% of the CPU consumption for both executions (i.e., whether or not the game is offloaded) under the two encoding qualities. This high CPU consumption represents a concern in mobile gaming. For the other modules that are not related to rendering, they represent less than 30% of the CPU consumption of the demanding games.



(a) Consumption per frame

(b) Consumption per Module

Figure 5.3: **CPU-time consumption per frame and module**

5.6.2 Network Communication

To test the network performance of our framework, we captured the network load incurred by offloading the game modules to the remote server. Packets were captured using *"Wireshark*[5]*"*. We limited the

[3] Interquartile Range (IQR) corresponds to the range of half of the scores around the median (the difference between the 75^{th} and the 25^{th} percentiles)

[4] The difference between the highest (90^{th}) and the lowest (10^{th}) score

[5] `https://www.wireshark.org/`

captures to 200 s.

We plot in Figure 5.4 the bitrate (in $bytes/s$) for the network traffic between the mobile device and the remote server for the two qualities encoding.

We captured both the uplink and downlink traffic between the client and the server. In the downlink direction, the packet size varies between 54 and 394 B, and the median is 88 B for both the fast and the good quality. On the uplink direction, the packets size varies between 60 and 162 B (60 and 228 B, respectively) with a median about 86.5 B (83.9 B, respectively) for the fast (good, respectively) quality. As stated above, only commands are streamed to the mobile device. Therefore, the variation in the bit rate depends on the number of offloaded GOs. Indeed, higher are the GOs offloaded to the remote server, the greater is the number of streamed commands, hence, the higher is the bit rate. Between the two qualities, there is also a variation in the bit rate. We believe that it is due to the difference in the time needed by the server to compute the GOs before streaming the commands. This time depends on the number and type of GOs, which *may* differ between the two qualities encoding.

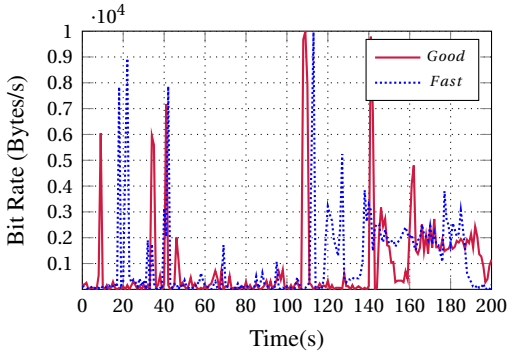
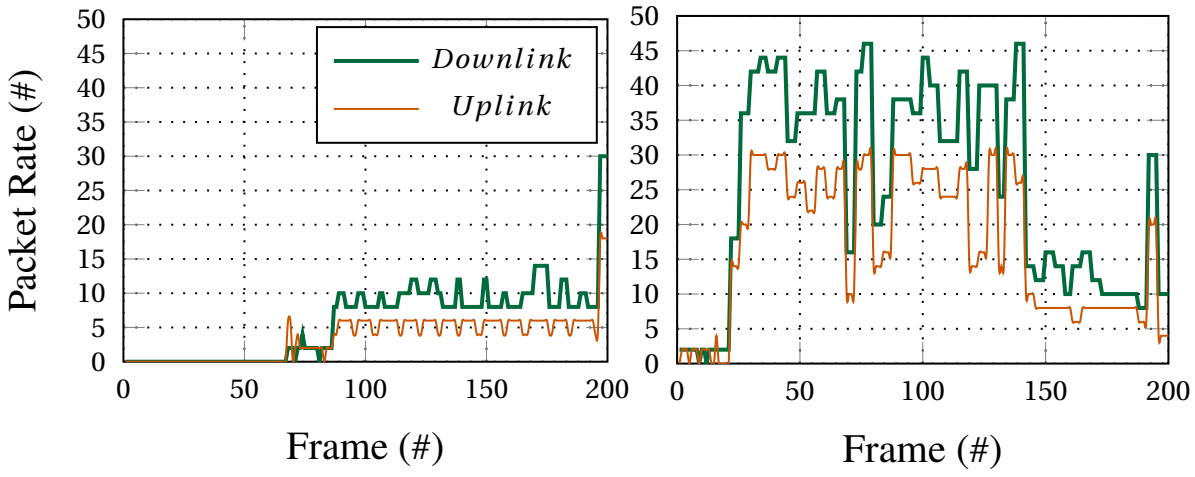


Figure 5.4: **Packets load per a tick of 1 second interval**

Figure 5.5a (5.5b, respectively) illustrates the number of packets captured on the uplink and downlink directions for each frame encoded with the good (fast, respectively) quality. We make three main observations:

- *Downlink rate is higher than the uplink rate*. The average rate for the fast (good, respectively) quality on the downlink direction is around 17.13 (3.94, respectively) $packets/frame$, while on the uplink direction, this average is about 11.51 (2.39, respectively) $packets/frame$. This is somehow obvious as our framework relies on the remote server to stream back rendering commands and update various modules. The *uplink* traffic represents only input commands and (N)PC variables.
- *The server follows the client pace*. Despite the powerful capabilities of the remote server, it has to follow the client pace to stream back the results, since the network manager synchronizes between them. Indeed, the server is triggered only when it receives an event (input commands or RPC) from the mobile device.

- *Server requested only on performance enhancement*. Both Figures 5.5a and 5.5b exhibit two behaviors; less and high network communications. When the heuristic estimates that no offloading gain can be achieved, then the whole game is computed on the mobile device, hence the server is in an idle state, that is to say, only few control packets are sent by the server. However, when a gain can be achieved, the network manager establishes a communication between the mobile device and the remote server to collaborate in the frame rendering.



(a) Good Quality

(b) Fast Qualiy

Figure 5.5: **Uplink and downlink packets rate per frame**

5.6.3 Responsiveness

To better understand how the client and server contributed in the frame generation, it is necessary to determine the interaction delay, which is consumed by several tasks: (i) capture of gamer input actions, (ii) transfer of command(s) to the remote server, (iii) execution of the m offloaded modules (om) on the server and the $n-m$ non-offloaded modules (nom) on the client (where n is the total number of modules), (v) stream OpenGL ES commands, and finally (vi) inject the commands in the graphic pipeline and render a frame. This overall interaction delay IDL is divided into three parts:

1. *Processing Delay*, PD, is the maximum time between local and remote execution. Local (remote, respectively) execution time is the sum of non-offloaded (offloaded, respectively) modules execution delays as given in Equation (5.2). We used the *visual studio* profiler to extract these delays.

$$PT = max\left(\sum_{i=1}^{n-m} t_i^{(nom)}, \sum_{i=1}^{m} t_i^{(om)}\right) \tag{5.2}$$

2. *Updating Delay*, UD, is the time spent to update *locally* modules inputs (such as the scripting module). We instrumented the code of the game to identify these delays. The UD is then, equal to

the sum of all the networked update times $t_i^{(uom)}$ for om_i, given by Equation (6.2).

$$UT = \sum_{i=1}^{m} t_i^{(uom)} \tag{5.3}$$

3. *Communication Delay*, CD, corresponds to the command streaming delay, input commands forwarding delay, and other control communication delay. It is the sum of the RTT and the time to send an amount of Q data as shown in Equation (5.4).

$$CT = \frac{(Packet\,Rate) \times (Packet\,Size)}{Bandwidth} + RTT \tag{5.4}$$

The ID is given by Equation (5.5). It corresponds to the average time obtained in Figure 5.3a.

$$RT = PT + UT + CT \tag{5.5}$$

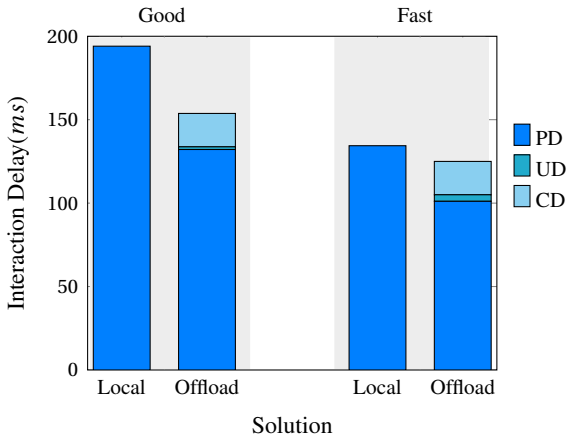


Figure 5.6: **Average of interaction Delay**

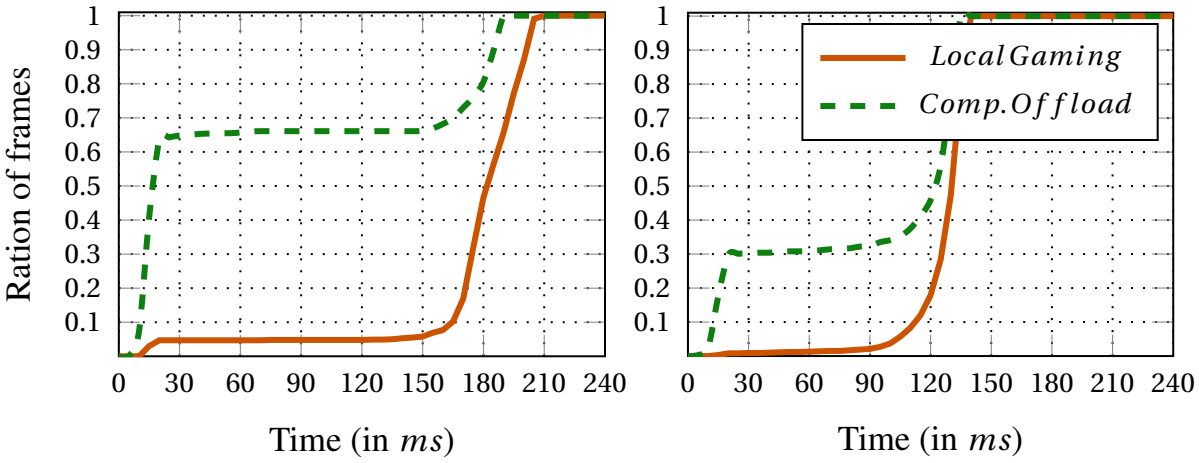
Figure 5.6 illustrates the average ID achieved under both the local and remote execution. We observe that: (1) our framework achieves a small UD, at most 3.91 ms (1.62 ms, respectively) for fast (good, respectively) quality. This time represents 3% (0.93%, respectively) of the ID. (2) The PD is 3× longer than what we expected, 105.79 ms and 151.48 ms for fast and good quality, respectively. Since we leverage powerful remote server, we hope closer performance to Cloud gaming solutions, as the consuming modules are computed on the remote server and only command packets transit over the network. We believe that this drawback is due to the injection of the OpenGL ES commands in the graphic pipeline.

5.6.4 Framerate

We conclude the performance section by summarizing the different results via the framerate performance.

Figures 5.7a and 5.7b depict the ratio of frames in a population of 1000 frames that are generated in less than $x\,ms$ for the good and fast quality, respectively. When using our heuristic, the game engine

generates more than 65% (30%, respectively) of frames in less than 33 ms for the good (fast, respectively) quality in comparison to the local execution, where the engine generates less than 10% of frames for the two qualities. Some frames are composed of several GOs that highly communicate through various modules which increase the interaction delay. Indeed, as the figures are showing, up to 35%, 70% of frames for the good, respectively fast quality need until 210 ms and 150 ms to be rendered. Our heuristic, in this case, offloads only a few clusters, as the communication cost is higher than the computational cost, or because no offloading gain is obtained for these frames.



(a) Good Quality

(b) Fast Qualiy

Figure 5.7: **CDF for frame generation**

5.7 Conclusion

In this chapter, we proposed a solution to play best-seller 3D games with high quality encoding, on powerless devices via offloading computation to a remote server. The concept is to identify modules involved by each GO in the game scene. Then, decide to offload them (as a whole) or not, depending on three main criteria: resource consumption, network communication, and code dependency. The mobile device and the server are synchronized through network manager, which orchestrates the offloading. The solution is scalable and adaptable to the network latency as only modules improving performance are offloaded. It supports mobility since network packets are automatically routed to the mobile device.

CHAPTER 6

MEC ORIENTED COMPUTATION OFFLOADING

6.1 Introduction

One of the 5G objectives is to reduce the network latency to $1\,ms$. This shall be possible only by bringing computation and storage resources closer to end-users. This has given rise to a novel concept, known as MEC or also Fog Computing. MEC is gaining lots of momentum, wherein the key idea is to empower mobile edge entities (e.g., radio access points such as eNodeBs and access gateways) with computation capabilities, allowing hosting and executing applications at the edge of mobile networks, rather than at a remote server in the Cloud or in the operator's core network domain. Combined with the 5G mobile access, which aims to drastically reduce the end-to-end latency, MEC will enable a plethora of novel mobile services that require low latency to access data or computation capabilities. Among the envisioned services are computation-offloading-driven applications.

Most of computation offloading works have been devised without considering MEC, since this concept is very recent. Moreover, they consider enforcing the offloading algorithms at the powerless device side, ignoring all information on the device network environment, especially varying radio quality. This can lead to dramatic situations, as the powerless devices may offload data even when the network conditions are bad (e.g., introducing high latency with the remote servers, not compatible with the devices application requirements). In this chapter, we use MEC as an enabler for low-latency computation offloading-based applications, not only by hosting the remote server on the ME, but also by devising a Mobile Edge (ME) service that continuously estimates the expected latency in order to feed the device decision to offload parts of an application's computing tasks. Indeed, according to the ETSI definition of MEC [73], the ME host is able to expose high level APIs to ME applications in order to provide real-time User Equipment (UE)-relevant network state information, making possible the prediction of UE network state (particularly the latency), and hence driving the UE decision of

when to offload, according to the network conditions. To this aim, we devised a framework that runs on three different entities: a ME application (hosted on ME host and capable to access UE-related radio information), a UE, and a server (hosted in the ME side to reduce end-to-end latency), enabling network-aware computation offloading of low-latency applications. The ME application is in charge of: (i) accepting or rejecting UE requests to offload or not, and (ii) predicting (using the Radio Network Information Service (RNIS) API) and sending latency information to the UE. The UE enforces all the steps composing the computation offloading process, while using the estimated RTT value obtained from the ME application. Finally, the server is in charge of executing the offloaded code and sending back the results to UE.

6.2 Background and Related Work

The chapter presents a novel approach for computation offloading in MEC environment. We designed an application hosted in the ME, which interacts with the MEC host through different services to; (i) give an approximation of latency between a UE and a server in MEC; (ii) check the network condition and resource availability for a subscriber demand, in one hand, and with the end-user to accept or refuse his requests for computation offloading, in the other hand. To better understand these contributions, we introduce hereafter the LTE and MEC concepts followed by the state-of-the-art computation offloading oriented to MEC.

6.2.1 LTE

6.2.1.1 evolved Packet System (EPS) Bearer

EPS Bearer (Bearer for short) [2] is an end-to-end tunnel between a UE and a P-GW , in which UE Internet Protocol (IP) packets are encapsulated inside General Packet Radio Service (GPRS) Tunneling Protocol (GTP) header. A bearer uniquely identifies packet flows with a common QoS parameters. Therefore, all packet flows mapped to the same bearer, follow the same forwarding-packet treatment (e.g., scheduling policy, queue management policy, and link-layer configuration). For each bearer, we define a QoS class and a UE's IP address. An end-to-end IP packet is encapsulated within a tunnel header containing a bearer identifier so that the network nodes can associate the packet with the correct QoS parameters.

Based on the QoS parameters, a bearer can be classified as either *default* or *dedicated* bearer, as shown in Figure 6.1.

- *Default Bearer*: It is set up when a UE attaches to an LTE network and it is kept as long as the UE retains that IP address. It does not guarantee the bit rate and it offers only best effort service. In 3GPP specifications, it is mandated that a default bearer is non-Guaranteed Bit Rate (GBR) bearer because it can remain established for a long period.

- *Dedicated Bearer*: It is dedicated to one or more specific traffic (e.g., Voice over IP (VOIP) and video). It acts as a secondary bearer on a top of the default bearer, with which it shares the IP address. The dedicated bearer can be either a GBR or non-GBR bearer. An operator distinguishes traffic flows mapped onto dedicated bearers and their QoS levels using the *Policy and Charging Resource Function (PCRF)*.

6.2.1.2 QoS Parameters

In the LTE network, the EPS bearer QoS is controlled by the LTE QoS parameters described in the following.

- Resource type: We distinguish two types of resources allocated to a bearer; *GBR* and *non-GBR* type.
  - *GBR bearer* guarantees bandwidth and avoid congestion-related packets losses. Therefore, a traffic carried by a GBR bearer conforms to GBR QoS parameter associated with the bearer. Consequently, GBR bearers are generally subject to admission control within the network when created or modified. GBR bearers are established *on demand* because they obstruct transmission resources and reserve them in an admission control function. GBR bearers are only established for dedicated bearers. The QoS Class Identifier (QCI) of a GBR bearer ranges from 1 to 4.
  - *non-GBR bearer* is a best effort type bearer and its bandwidth is not guaranteed. A service using a non-GBR bearer must be prepared to experience congestion-related packet loss. A default bearer is always a non-GBR bearer, whereas a dedicated bearer can be either GBR or non-GBR. Non-GBR bearer does not block any network specific traffic or transmission resources. It can remain established for a long time, its QCI ranges from 5 to 9.
- QoS parameters:
  - *QCI* [3] is a scalar ranging from 1 to 9 to indicate *nine* different QoS performance characteristics of each IP packet. The QCI is used within the access network as a reference to node-specific parameters that give details of how an LTE node handles packet forwarding (e.g., scheduling weight, admission thresholds, and queue threshold). Operators deploy pre-configured nodes to handle packet forwarding according to QCI value. QCI values are standardized to reference specific QoS characteristics, and each QCI is associated with standardized performance characteristics that describe the packet-forwarding treatment that the bearer traffic receives in terms of *resource type* (GBR or non-GBR), *priority* (1 to 9), *packet delay budget* (from $50\,ms$ to $300\,ms$), and *packet error loss rate* (from 10^{-6} to 10^{-2}).
  - *Allocation and Retention Priority (ARP)* is a scalar ranging from 1 to 15, with 1 being the highest level of priority. an ARP specifies the control-plane treatment that a bearer receives.

More precisely, it allows an LTE entity (e.g., P-GW, S-GW, or eNB) to decide whether a bearer establishment or modification request can be accepted or be rejected, and further, which bearer to release during network congestion.

– *Maximum Bit Rate (MBR)* and *GBR* are used only for GBR bearers. GBR and MBR define respectively, the minimum bit rate to be guaranteed by LTE networks and the maximum bit rate that a traffic in a bearer may not exceed. Any packets arriving at the bearer after the specified MBR is exceeded will be discarded.

– *Aggregate Maximum Bit Rate (AMBR)* is used to limit the total amount of bit rate consumed by a single subscriber. It is not defined per bearer, but rather per the entire non-GBR EPS bearer. In 3GPP specifications, we distinguish two AMBR parameters: (i) *UE-AMBR*, which is defined per subscriber and it is known by both the gateway and the RAN; (ii) *APN-AMBR*, which is also defined per subscriber, but only known to the gateway. The UE-AMBR and APN-AMBR are defined for an aggregate non-GBR bearers. These AMBR parameters are defined separately for uplink (UL) and dowlink (DL) direction. Therefore, there is four values for AMBR; (i) UL UE-AMBR; (ii) DL UE-AMBR; (iii) UL APN-AMBR; and (iv) DL APN-AMBR. The bit rate consumed by a GBR bearer is included in these AMBR values.

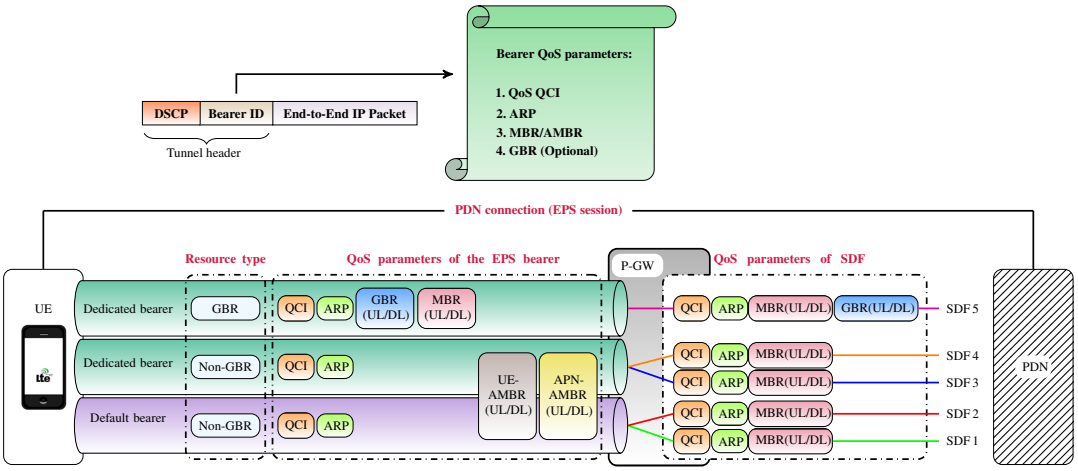


Figure 6.1: **Bearers with associated QoS parameters**

6.2.1.3 Scheduling in LTE

In LTE mobile access, the eNodeB's scheduler in the Medium Access Control (MAC) layer allocates available radio resources among different UEs in a cell, based on priority. LTE uses Orthogonal Frequency Division Multiplexing (OFDMA) for downlink transmission, mapped on a *time-frequency* resource grid. Data are allocated to UEs in terms of Resource Blocks (RBs). In *time* domain, a RB

length is equal to one slot[1](i.e., 0.5 ms), while in the *frequency* domain, the length of a RB is 180 KHz organized in 12 sub-carriers of 15 KHz spacing.

The used scheduling method largely impacts the throughput of individual users as well as throughput of the cell. Different scheduling methods have been proposed in the literature to deal with fairness and throughput.

- *Round Robin (RR)* [173]: Resources are furnished cyclically to users without considering channel conditions. That is, the RBs are assigned equally among users (i.e.,in turn, one after another) regardless of CQI. The scheduler is aiming at giving the best fairness by providing an equal share of packet transmission time to each user. Thus, UEs are equally scheduled. However, throughput performance may degrade significantly due to the non consideration of the reported instantaneous downlink Signal to Noise Ratio (SNR) values in determining the number of transmitted bits.
- *Proportional Fair (PF)* [159]: The algorithm is balancing between throughput and fairness among all UEs. PF aims to maximize, over all feasible scheduling rules, the utility function $\sum_i logR_i$, where R_i is the long-term service-rate of user i, by serving the user with the highest ratio $r_i(t)/R_i(t)$ at each time slot t; $r_i(t)$ and $R_i(t)$ represent respectively, the quality channel-rate and the current average service-rate. That is, the scheduler tries to maximize the total throughput while, at the same time, provides all users at least a minimal level of service.
- Best Channel Quality Indicator (CQI) [253]: In this approach, the resource blocks are assigned to a user with the best radio link conditions. To perform a scheduling, an eNodeB sends a reference signal to a UE in order to measure its CQI. The UE generates the CQI reports[2] and fed back them periodically, to the eNodeB in a quantized form. A higher CQI means better channel condition. The scheduler can increase the cell capacity at the expense of fairness. That is, UEs that are far away from the eNodeB are unlikely to be scheduled.

6.2.2 MEC

MEC is an emerging ecosystem that provides an IT and Cloud computing capabilities (i.e., storage and computational resources) at the RAN edge in a near vicinity to end users. Besides the supplied resources, MEC offers real-time access to radio and network analytics and efficiently uses the mobile backhaul and core networks. MEC allows developing a plethora of new applications and services, consumed in an ultra-low latency and high bandwidth. The ecosystem is useful for service providers to introduce new services and differentiate costumers portfolio, by collecting information regarding costumers content, location, and interests. MEC offers an open radio network edge platform, facilitating multi-service and multi-tenancy. That is, authorized third-parties are allowed to use storage and processing capabilities and introduce on-demand, in flexible manner new businesses. MEC should enable a secure Cloud

[1]Number of symbols in one slot is 7 (6, respectively) for normal cyclic prefix (extended cyclic prefix, respectively) in case of 15 Khz sub-carrier spacing.

[2]Contain the signal-to-noise and -interference ratio (SINR) value measured by the UE.

platform architecture and provide APIs to dynamically share and use the MEC platform, easily installing and modifying new services, and efficiently interacting with the RAN, e.g., being able to retrieve RAN-relevant information.

6.2.2.1 MEC Architecture

MEC is deemed as a critical technology to enable the transition toward 5^{th} Generation (5G). Behind MEC standard, ETSI would mainly ease the deployment of UE applications requiring short latency access to remote servers. ETSI proposes a complete reference architecture that defines functional blocks and interfaces to describe the MEC system [73–75].

The ETSI MEC framework describes three levels of entities and functions; *mobile edge system*, *mobile edge host*, and *network* level entities. Figure 6.2 portrays a simplified version of the MEC architecture as defined by ETSI [73]. This architecture concentrates on the mobile edge system level and on the mobile edge host level excluding the network level. The ME management system comprises the *ME orchestrator*, the *ME platform manager*, and the *Virtualization Infrastructure Manager (VIM)*. The *ME orchestrator* has the view on the whole ME system, as it maintains the information about all the deployed ME hosts, the services and resources available in each host, the instantiated ME applications and the network's topology. The orchestrator is also responsible about installing ME applications, checking their authenticity, and validating the associated policies. It has a reference point with the operator's Operating Support System (OSS), which is the highest level management entity in a mobile network. The *ME host* is the logical entity that contains the ME platform and the virtualization infrastructure on which the ME applications run. The *ME platform* contains a set of baseline functions that enable ME applications to run on a particular host, to discover and consume ME services or to advertise and provide them through the service registry. The ME platform is also responsible for enforcing the traffic rules to route the data packets to/from the ME applications, and to maintain a Domain Name System (DNS) subsystem to discover the ME applications. *ME applications* run on the ME host as virtual instances (i.e., VMs or containers) and are designed to consume and/or provide *ME services*. The latter provide high level APIs that expose UEs status (radio and context information), which will be then used by the ME applications to optimize a registered service (e.g., offload computation).

6.2.2.2 ME Host Services

The MEC application development framework [35] implemented in the ME host provides services and APIs for high-layer MEC applications. The framework is composed of four type of services highlighted through Figure 6.3.

- **Common services**. They are central in the MEC host as they facilitate the usage of the real-time network and radio information. In particular, the RNIS APIs, which exposes the following RAN information:

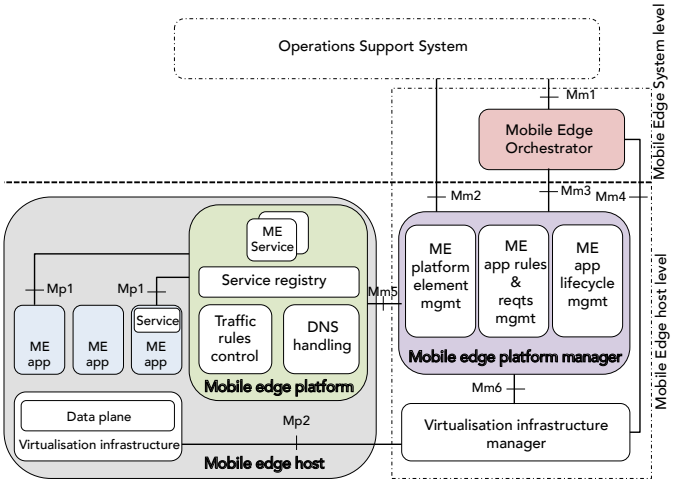


Figure 6.2: **(Simplified) MEC architecture**

- *Radio quality indicators*: related to UE/eNodeB layer1/layer2 parameters. It includes up-to-date information regarding the configuration and status of UEs and the access network. We may mention the following: UE's configuration information (e.g., Public Land Mobile Network Identifier (PLMN ID), Cell-Radio Network Temporary Identifier (C-RNTI), downlink/uplink (DL/UL) bandwidth); UE status information (e.g., Global Navigation Satellite System (GNSS)); eNodeB configuration information (DL/UL radio bearer configuration, tracking area code, PLMN identity); eNodeB status information (GNSS, DL/UL scheduling information, number of active UE).
- *Control-plane interface*: Exposes information on UE/eNodeB layer3, S1/X2 interface messages, used for network control. We may mention: UE status information (mobility state, mobility history report (X2); radio link failure report); eNodeB status information (including, physical RB usage per traffic class).
- *Data-plane interface*: It provides information on the X2-U and S1-U interfaces, such as UE configuration information (EPS bearer identity, bearer type (default or dedicated), bearer context (CQI, ARP), bearer bit rate (GBR or MBR)), UE status information (QCI, Channel State Information (CSI)), and network status information (aggregated physical RB usage, delay jitter of specific QCI).

- **MEC services**. Provide a common high-level APIs to the different MEC applications and use cases. These services include Key Performance Indicators (KPIs) evaluation and traffic profiling, positioning, IP and named data services, network status and configuration, analytic, and event capture.
- **Support services**. Provide common, specific functionality to most MEC services. They represent a basic platform services, that other more elaborated services can use. These services include but

are not limited to; communication service, discovery and registry, policy and charging, monitoring, authentication, authorization, accounting, and SLA.

- **Platform services**. Provide physical and virtual resources through an orchestrator such as OpenStack. The provided resources include the compute resources (CPU, and GPU), storage (e.g., RAM, and disc), network resources (vNIC), and I/O. SDN and Network Function Virtualization (NFV) operations belong to this service.

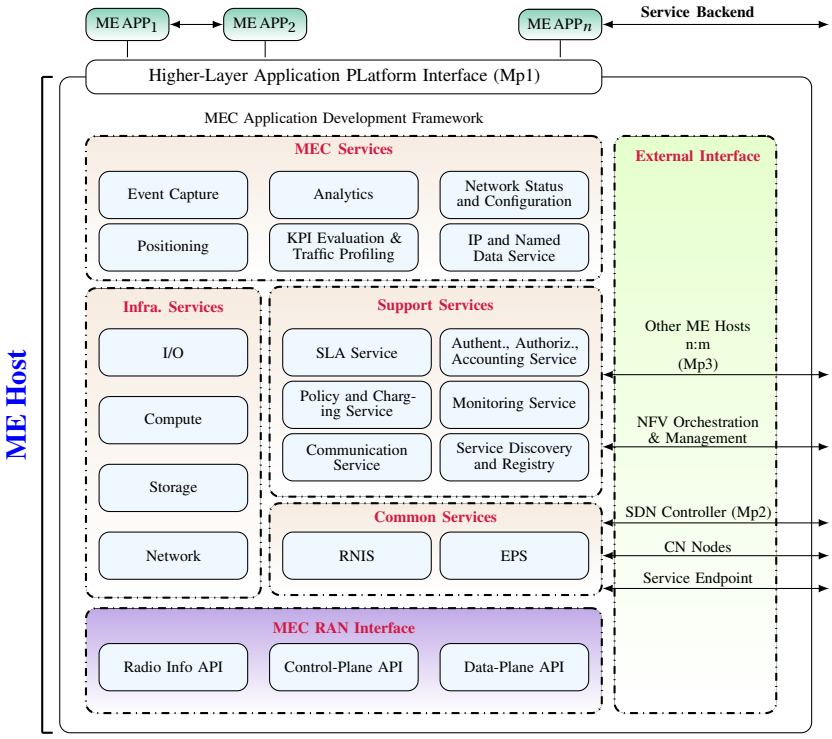


Figure 6.3: **ME host architecture**

6.2.2.3 MEC oriented for computation offloading

Few studies have been proposed to use MEC as an enabler for computation offloading mechanism. For instance, in [321], authors studied the consumed energy in a computation offloading context for MEC in 5G heterogeneous networks. The authors have formulated an optimization problem that minimizes the energy consumption of the offloading scheme, which takes into consideration the energy consumption of both task computing and data transmission. The solution incorporates the multi-access characteristics of 5G networks. Therefore, a joint optimization problem was addressed to optimize the offloading and radio resource allocation under latency constraints. In [183], authors have considered a MEC environment with several UEs, wherein tasks arrive in stochastic fashion. A trade-off is investigated between power consumption on mobile devices and execution delay for computation tasks. To this aim,

a power consumption minimization problem with task buffer stability constraints was formulated. The authors have developed an online algorithm, based on Lyapunov optimization, to derive the optimal CPU-cycle frequencies as well as the transmission power and bandwidth allocation vector for computation offloading. Authors in [293] proposed a framework-based computation offloading with interference management oriented to wireless cellular network with MEC. Computation offloading decision, Physical Resource Block (PRB) allocation, and MEC computation resource allocation are jointly addressed in as optimization problems. The offloading decision is made according to the local and offloading overheads experienced by the entire UEs and MEC server, respectively. The PRB allocation problem is based on the offloading decision and it is solved by the well-know graph coloring algorithm in graph theory. A multi-mobile-users MEC system was considered in [327], where multiple UEs request for computation offloading to a MEC server. To optimize the energy consumption, the authors have proposed to joint between optimal offloading selection, radio resource allocation, and computational resource allocation. An integer nonlinear programming model was drawn with the objective function to minimize the energy consumption, subject to latency constraint. To solve the nonlinear model, the authors proposed a reformulation-linearization-technique-based Branch-and-Bound method with an exponential complexity and a Gini coefficient-based greedy heuristic, which has a polynomial complexity. In [169], a power-constrained delay minimization problem for computation offloading based on Markov decision processes has been proposed and adapted to the MEC context. The authors have considered a mobile device running computation-intensive and delay-sensitive applications. The device is composed of a task buffer, a transmission unit, and a processing unit. An algorithm has been designed based on the average delay of each task and the average power consumption at the mobile device to solve the power-constrained delay model and find the optimal scheduling. An energy-latency trade-off for energy-aware offloading in MEC networks was proposed in [320]. The scheme jointly optimizes communication and computation resource allocation under the limited energy and sensitive latency. The authors have considered the computation offloading in single and multi-cell MEC networks scenarios. The residual energy of UEs' battery was introduced into definition of weighting factors for energy consumption and latency subjectively defined by mobile users. An iterative search algorithm combining interior penalty function method with the D.C. programming (difference of two convex functions/sets) was proposed to find the optimal solution. A multi-user resource allocation for MEC has been also proposed in [315]. The model is formulated as a convex optimization problem to minimize the mobile energy consumption considering the computation overhead and the capacity of the MEC. The model derives an offloading priority for each user according to its channel gain and energy consumption. A low priority derives a minimum offloading, while a high priority performs a complete offloading. Lastly, an extended study of computation offloading in a multi-cell environment was proposed in [247], wherein the authors considered a Multiple Inputs Multiple Outputs (MIMO) multi-cell system, with multiple users requests for computation offloading. The authors formulated the problem using a joint optimization of the radio and the computational resources for computation offloading in a dense deployment, with the presence of

inter-cell interference.

Clearly, none of these proposed solutions use the MEC services to drive the decision to offload or not. They all rely on local device's information to take such a decision. Our proposed framework overtakes this limitation by highly interacting with the MEC service to better predict UE's network quality of service, and thus considering up-to-date information to take offloading decision.

6.3 Proposed Framework

6.3.1 General description

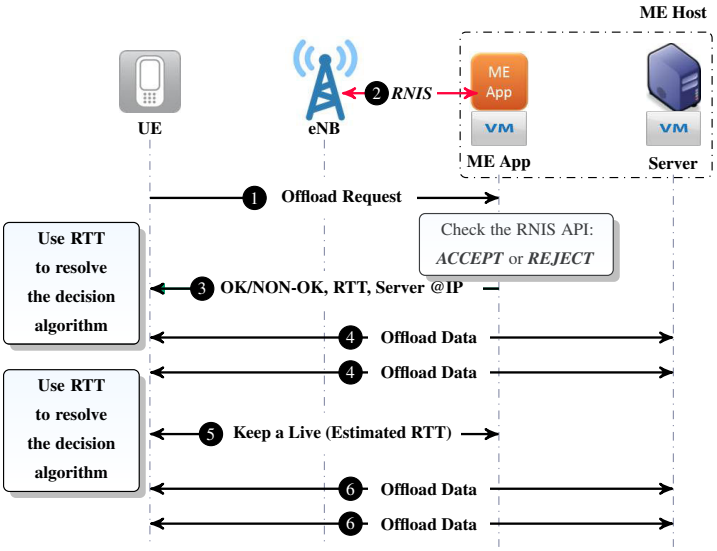


Figure 6.4: **Global overview of the proposed framework**

The key idea of the proposed solution is to enable UEs to offload computation tasks to a remote server located near to the ME host. As stated before, MEC allows the reduction of the end to end latency, which may ease the computation offloading process. Unlike the existing solutions, where UEs take locally, without the remote server help, the decision to offload or not a part of the computation task, in this work we rely on the MEC architecture, to not only host the remote server, but also drive the UE decision to offload or not a computation task. To this end, we propose to dedicate a ME application for computation offloading (*MEACOF*[3]), on top of the ME host, to steer the UE decision to offload or not. The ME application is in charge of initiating the computation offloading, using up-to-date information available within the RNIS API, and help the UE in taking the decision to offload or not.

One of the inputs usually considered to decide whether to offload computation tasks, is the latency between the remote server (hosted in the Cloud) and UE. In mobile networks, the latency value is impacted by different parameters, which depend on both the core network congestion level or the number

[3]Through this chapter, we use MEACOF and ME application interchangeably

of hops, and the wireless channel access. Note that hosting a server in the mobile edge to run the offloaded code prevents core network delay issue. Indeed, any server located near to an eNodeB is being reachable within a one hop connection. Therefore, only the wireless channel access may have impact on the latency. This latter is impacted by RBs allocated among different UEs in the same cell according to the priority of each UE. For the PF, Best CQI, and weighted RR schedulers, the higher the UE priority, the more often the UE is scheduled, so shorter the latency to access the wireless channel will be. The priority of each UE depends on his subscribed QoS parameters (refers to Section 6.2.1).

In another way, if a UE has a highly bad channel conditions it may not be scheduled often, which may increase the latency. In addition, depending on the number of active UEs in a cell as well as the scheduler policy for a UE in that cell, the packets towards that UE (or from that UE to a MEC server) may experience delay. Indeed, on the downlink (uplink, respectively) direction, if the number of active UEs in the cell is high and a UE is using a non-GBR type bearer (a default bearer or a dedicated bearer with no GBR), the packets towards that UE (from that UE to a MEC server, respectively) might experience high delays at the eNodeB downlink queue (UE uplink queue).

Approximating the Round Trip Time (RTT) value in the wireless channel may be challenging as many types of information are required, such as the number of active UEs in the cell, the channel quality of the UE (CQI and CSI), and the downlink and uplink bandwidth. But knowing that the ME application has access to these information via the RNIS API, it is possible to approximate the RTT and hence help the UE to decide to offload or not a part of the computation tasks. A global view of the proposed solution is shown in Figure 6.4. At the beginning, the ME application registers to use the RNIS API for specific UEs through the *service registry*, shown in Figure 6.2. Once a UE sends a request to offload data to a remote server (1), the ME application checks the RNIS API (2). If the ME application considers that it is better to executes the application locally on the UE, then it sends a reject (*NON-OK message*); otherwise it sends an accept (*OK message*) that includes the RTT approximation and the IP address of the ME server to connect to (3). The RTT value will be used upstream with energy and execution time to obtain the remote and local partitions, then offload the remote partition to the server having the IP address included in the OK message (4). The ME application approximates the RTT periodically and sends a *Keep-to-Live message* including the estimated RTT (5). The UE synchronized with the ME application, computes the partitioning algorithm at each period (i.e., use the new RTT to find the new partitions) and offload the remote partition (6). More details are given hereafter.

6.3.2 ME application Side

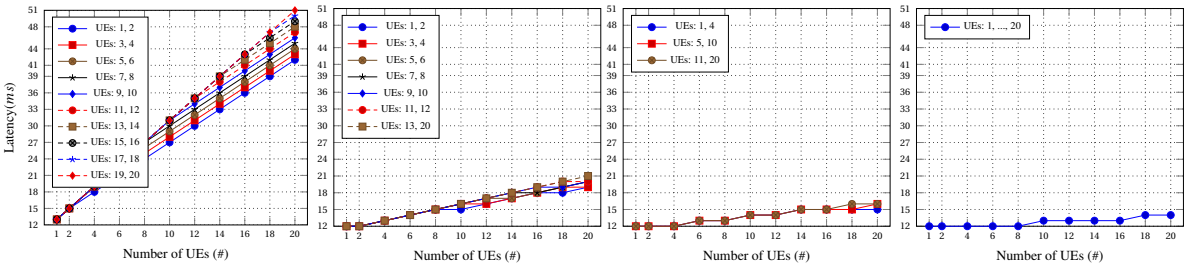
Once receiving the first request from a UE asking to offload data, the ME application, through different services gather several information to: (1) give an approximation of the RTT, (2) check the resource availability on the MEC, and (3) check the network conditions to accept or not subscribers offloading requests. Diagram in Figure 6.8 summarizes these actions, which are detailed in the following.
The minimum quality of the computation offloading service is guaranteed via the arguments defined by:

Arguments: Quality (GBR, MBR, loss rate, and priority), protocol configuration (IP, named service), and the needed IT resources for the computation offloading service.

Figure 6.7 depicts the involved services for the MEACOF. The support service translates the MEACOF arguments into RNIS and EPS related arguments. These two common services take the argument values translated as inputs by the support services and look up in their internal data bases and output the parameters to the MEC services: Network Status and Configuration as well as IP and Named Data Services. The Network Status and Configuration applies the required QoS parameters to establish a bearer for the computation offloading service between the UE and the server hosted in the MEC. The IP and Named Data Services adds the computation offloading header with specified protocol configuration for the traffic routing performed by EPS. Some KPI requirements are translated into platform services related requirements (such as VM profiles) and through the VIM (OpenStack for example), these services check resource availability for compute, network, and storage.

1. RTT approximation. Many works have tried to model the access latency in LTE radio access. Among them, the authors in [15] proposed to rely on the Remaining Block Signal (RBS), which is sent periodically by the UE to the eNB in order to estimate the access delay. Though this solution shows a good approximation, it is difficult to use it at the MEACOF, since the RBS will arrive delayed to the ME application, leading to wrong approximations. Our solution is based on the UL scheduling report obtained via the RNIS API. Indeed, the UL Scheduling information indicates the PRB assignment per UE. The ME application will gather the UL information during a period of time, noted T, and analyze it to approximate the delay. To do so, we perform some simulations to provide an LTE environment in which several UEs are connected to an eNodeB using different bandwidths. The simulation details are provided in Section 6.4.1.

To better understand the eNodeB's scheduler behaviour, we limited the communication to *one* packet par UEs and only on the uplink direction. Results are shown in Figures 6.5, 6.6.



(a) **Bandwidth =** $1.4\,MHz$ (b) **Bandwidth =** $5\,MHz$ (c) **Bandwidth =** $10\,MHz$ (d) **Bandwidth =** $20\,MHz$

Figure 6.5: **Latency values for 1 packet when eNB using RR scheduling**

Figure 6.5 presents the results in term of network latency (in ms) between the UEs and eNodeB to send one UDP packet for four bandwidth capacities: $1.4\,MHz$, $5\,MHz$, $10\,MHz$, and $20\,MHz$. The

eNodeB schedules one to twenty UEs in a Round Robin fashion for each bandwidth value. Figure 6.6 shows the size (in *Bytes*) of the RBs allocated to the UEs at each Transmission Time Interval (TTI).

The obtained results confirm that predicting the packet delay depends on four main factors: the number of UEs in the cell, the bandwidth capacity, the eNodeB scheduler, and the packet size.

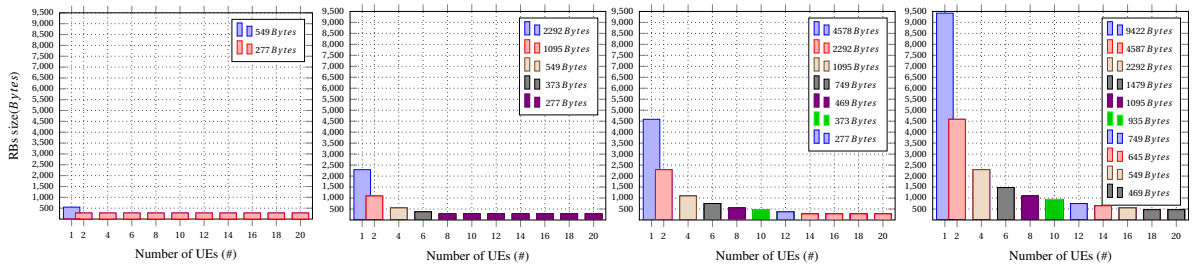
Impact of the number of UEs. The packet delay is proportional to the number of UEs in the cell. For instance, in Figure 6.5a, when *twenty* UEs are interacting with the eNodeB in the same cell, the delay for the packets sent by the UE #19 and #20 take $51\,ms$ to arrive to the eNodeB. However, in case of one UE is interacting with the eNodeB in the cell, the packet delay is $13\,ms$. We observe the same impact in the other cases (Figure 6.5b, 6.5c, and 6.5d), where the network latency is increasing with the number of active UEs in the cell. However, as the used bandwidths are larger than the one used in Figure 6.5a, the latency is reduced by a factor ranging from *two* to *four* for the UEs #19 and #20. The reasons behind this impact are the following. Within a cell, increasing the number of UEs, will increase the communication, that is the number of transmitted packets to the eNodeB. Therefore, the UEs concurrently share the bandwidth to transmit their packets. The scheduler in the eNodeB should allocate the bandwidth resource (in term of RBs) with a certain policy. Therefore, each UE packet is segmented with respect to fairness.

Impact of the bandwidth capacity. The bandwidth has a direct impact on the delay; the higher is the bandwidth, the larger number of RBs can be allocated, so shorter the latency will be (with respect to the number of UEs in the cell). The bandwidth in term of RBs varies from $6\,RBs$ to $100\,RBs$. The smallest resource capacity that can be allocated for a UE is the RB ($180\,KHz$ organized in 12 sub-carriers of $15\,KHz$ spacing). In case of $1.4\,MHz$ (i.e., Figure 6.5a), The bandwidth carries $6RBs$. Therefore, in the best case, only *six* UEs can be scheduled simultaneously, the other UEs should wait for other TTIs to be seen allocated RBs, and hence increase the delay with at least $1\,ms$ (in case of resource allocation in the next TTI). In the other hand, when the bandwidth provides $100\,RBs$ (i.e., capacity of $20\,MHz$), more then six UEs are packaged within the same TTI to send their packets. Indeed, the entire *twenty* UEs are scheduled in the same TTI.

Impact of the eNodeB's scheduler. The scheduling policy matter in network delay. Indeed, according to the scheduler policy in the eNodeB, a UE may take a long time to be scheduled, due to its *poor* modulation, because this latter depends on the radio conditions of the UE. As described in Section 6.2.1.3, the RR scheduler allocates RBs equally and periodically among the entire UEs, without considering the channel conditions, while the PF scheduler is serving UEs with the highest ratio between the quality channel-rate and the current average service-rate. In addition, CQI scheduler allocates resources according to users with the best radio link conditions. Therefore, each scheduler has its own algorithm to allocate resources, which promote a group of users, then reduce their network latency, and disadvantage other users, hence increase their network latency.

Impact of the packet size. The packet size has also an impact on the network delay that we illustrate via two examples: Voice over LTE (VoLTE) call packet, and a UDP packet. In case of VoLTE call, the

size of a packet depends on the used Codec for the voice. If we consider an Adaptive Multi Rate-Wide Band Codec (AMR-12.65), every 20 ms are generated 253 $bits$. To deliver a voice sample to a UE, we need to add other protocol headers such as Real-Time Transport Protocol (RTP) header (typically 12 $bytes$), a UDP header (8 $bytes$), and an IP header (40 $bytes$). Hence, the entire VoLTE packet length is equal to 733 $bits$ generated every 20 ms. For a UDP packet is about 1024 $bytes$ (8 $bytes$ for UDP header, 40 $bytes$ for IP header, and 976 $bytes$ for the payload). We are interested here into the number of radio resources needed to transmit in the air for the VoLTE call and the UDP packets. In LTE, a RB is composed of 12×7 resource elements (REs) (for a normal Cyclic Prefix). Based on the LTE downlink modulation, a RE carries respectively, 2, 4 or 6 $bits$ for Quadrature Phase-Shift Keying (QPSK), 16 Quadrature Amplitude Modulation (QAM), and 64 QAM modulation scheme. The modulation scheme is chosen by the eNodeB with the respect to UE CQI. For example, a UE using CQI #15 (i.e., good channel condition), the eNodeB can use a modulation of 64 QAM. Therefore, each RE holds 6 $bits$. Accordingly, a RB can carry $168 \times 6 = 504$ $bits$. When the Robust Header Compression (RoHC) method is used for the VoLTE call, the VoLTE packet is compacted in 300 $bits$. Knowing that a RB is the smallest resource allocated per UE by the eNodeB, the VoLTE call needs *one* RB, while the UDP packets needs *sixteen* RBs. When the bandwidth cannot carries the entire RBs needed for a packet, the latter is fragmented into several frames, and when the eNodeB is larger than the packet size, padding information is added to the packet. Figure 6.6 summarizes the number of RB (in bytes) needed to transmit one UPD packet, for different bandwidth capacities.



(a) **Bandwidth =** 1.4 MHz (b) **Bandwidth =** 5 MHz (c) **Bandwidth =** 10 MHz (d) **Bandwidth =** 20 MHz

Figure 6.6: **RBs size (in Bytes) for 1 UL RR scheduling**

To sum up: The number of RBs composing a packet, the bandwidth capacity, the number of UEs in the cell, and the scheduling policy impact the network delay. It is difficult to predict the network latency using all these information in complex calculation. What we propose to do is to use the RB and the scheduling allocation calculated by the eNodeB for each UE. To obtain these information, the eNodeB takes into consideration, the number of UEs in the cell, the scheduling policy, and the bandwidth. In this case, we only need to use these two information (RBs and scheduling allocation) gathered into the scheduling report, and obtained through the RNIS API, besides the size of packets in term of RBs.

We know that the TTI (or the smallest scheduling interval) in LTE is 1 ms. If we fix T to 1 s, then each period will include 1000 UL scheduling samples to be analyzed. If the UE has been scheduled X

times during T, we can estimate the average scheduling delay during the period T, as T/X. In addition, the average of RBs that have been allocated for a UE during the period T is given by $\lceil \sum_{i=1}^{T} RB_{alloc}/X \rceil$. The average access delay noted by D is given by:

$$D = \frac{T}{X} \times \frac{PacketSize(RBs)}{\lceil \frac{\sum_{i=1}^{T} RB_{alloc}}{X} \rceil} + 10 \tag{6.1}$$

Of course this calculation represents an average, which may not lead to an exact approximation, but at least it can be used easily to estimate the UE's packet delay, requiring only the UL scheduling information available through the RNIS API. We believe that the $10\,ms$ in the equation corresponds to the frame duration.

It is well known that the RTT could be approximated by multiplying the access delay by two, as the DL latency could not exceed the UL latency but would somehow be equivalent. Moreover, the backhaul latency may be omitted, as the server is located at the ME host. Then, the approximated RTT is equal to $2 \times D$. To avoid a sudden fluctuation of the estimated RTT, an Exponential Weighting Moving Average (EWMA) technique is used. The final value of the RTT, to be sent to the UE periodically is as follows:

$$RTT_{est} = \alpha \times RTT_{prec} + (1 - \alpha) \times RTT_{cur} \tag{6.2}$$

where $\alpha \in [0, 1]$, and RTT_{prec} and RTT_{cur} represent the most recent RTT estimate and RTT sample respectively.

2. Resource availability. The ME application accesses to the infrastructure service to check if sufficient resources can be reserved for the UE to compute the offloading code regarding to the UE requested requirements. Indeed, if the UE requirements are computation-intensive then, the ME application will reserve sufficient CPU and/or GPU if available. If the requirements are more memory-intensive than computation then, the ME application will reserve enough space (including disc, RAM, and cache). Therefore, the ME application will request for a VM creation in the nearest server, which will be connected to the UE.

3. Network conditions. From the RNIS and EPS API, the ME application knows the number of active UEs in the cell, the UE bearer type and the CQI of the UE. Obviously, if the UE's CQI is bad, then it is better to execute the code locally as UE packets may experience high delays (due to the low modulation throughput), which may degrade the QoE and the QoS. Furthermore, form the EPS API, the ME application identifies which UEs are using GBR and Non-GBR bearers. if the UE is using a non-GBR bearer and the number of active UE is high, then the offloading request is rejected. Even if the UE has a dedicated bearer, the ME application should check the associated QCI in order to evaluate the packet delay budget allowed to this bearer.

If the ME application estimates that the channel conditions are good or the UE has a dedicated bearer that guarantees low delay access and there is sufficient resources on the MEC, the UE is authorized to

offload. The decision along with the remote IP address and an RTT approximation (i.e., if the offload is accepted) are included in the response of the ME application to UE. The initial RTT value could be the packet delay budget associated with the UE bearer type (i.e., maximum tolerated latency). Then periodically, the ME application will further approximate the RTT according to the RNIS information and provide it to the UE.

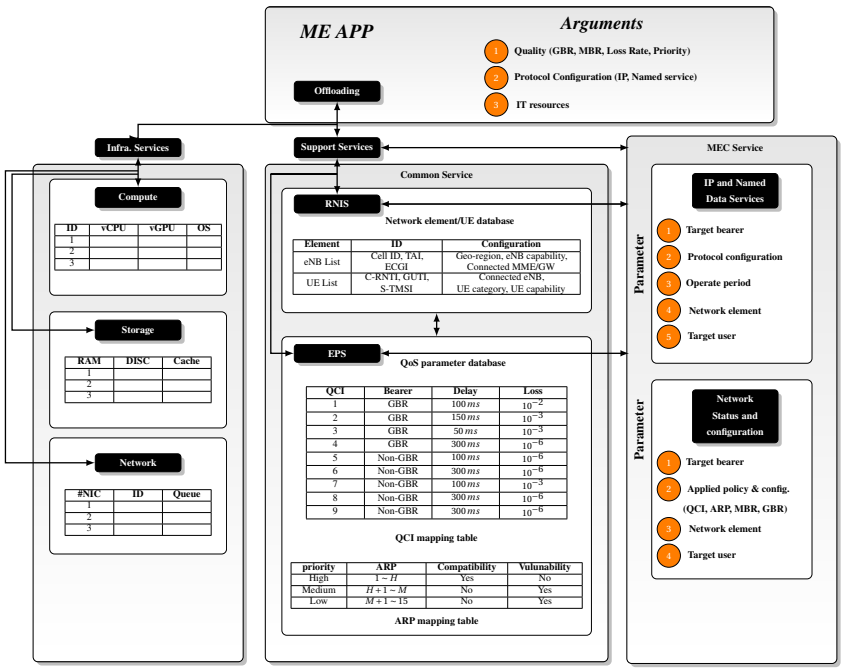


Figure 6.7: **ME application (interconnection with the different services)**

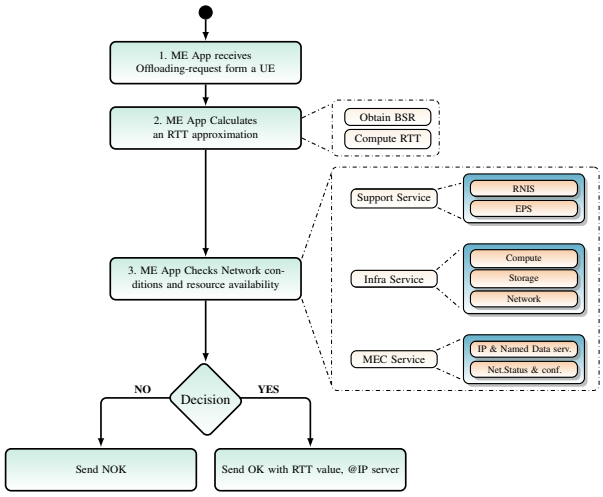


Figure 6.8: **ME application flowchart**

6.3.3 UE Side

The UE's framework, represented in Figure 6.9, in addition to profiling, scheduling, and proxy, it includes the following functionalities:

- **Modelling -** Modellizes applications into a valued graph.
- **Partitioning -** Solves the optimization problem via graph partitioning algorithms.
- **Monitoring -** Monitors resources on the UE (e.g., remaining energy, CPU load), and transmits periodically, these values with the RTT estimation to the Partitioning module.
- **Patching -** introduces keyword *"non-transferable"* for classes with a code dependency constraint [59, 94, 216, 217, 220].

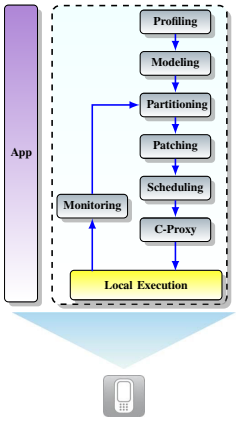


Figure 6.9: **UE framework**

The process at the UE side is following the computation offloading described in Section 2.2. We start by profiling the application to estimate the resource consumption, then we modellize the application via a valued graph. Each vertex represents a software component with a chosen granularity. At this stage, the graph may have a large size. Therefore, we propose to reduce it by grouping vertices that highly communicate to create clusters of components. Each time a ME application decision is received (OK message), the partitioning module is launched in order to find the local and remote partitions. The remote partition is then offloaded through a bearer established with the MEC server. Communications are based on RPC mechanism, done between the C-Proxy and S-Proxy. The diagram in Figure 6.10 highlights the steps achieved by the UE to launch a computation offloading with a server hosted in the MEC. These different steps are thorough in the following paragraphs.

1. Profiling and Modelization. Basically, We assume that the application candidate for computation offloading is composed by n classes. That is, the granularity level of offloading is the *class*. The application is represented using a *valued and locality-labeled graph (VLG)* with n vertices. Each vertex in the graph corresponds to one class in the application; it is labeled with a binary value to indicate

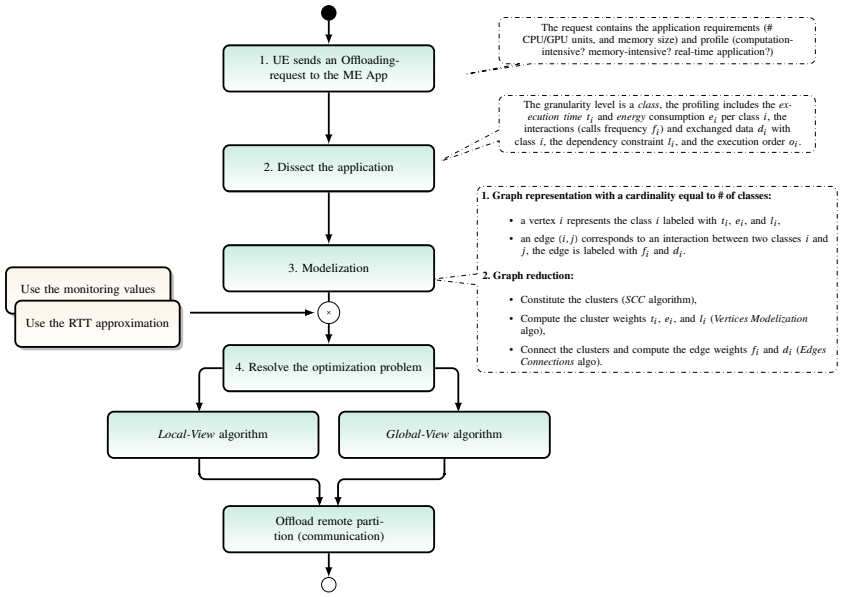


Figure 6.10: **UE flowchart**

whether the component can be offloaded or not, according to the dependency constraint. Let us call this label *locality l*.

We used the execution time on the UE and the energy consumption to assign values to vertices. Each edge in the graph represents the interactions between two classes in the application and is weighted with the frequency of calls as well as the size of the exchanged data. The problem consists in identifying which classes of the application could be offloaded aiming at improving the response time and saving energy on the UE device. To this end, we propose to use the graph partitioning technique to obtain a minimum cut. Stemming from the fact that graph partitioning is an NP-hard problem (thus, its solution is prohibitive computationally), we propose to reduce the graph size by clustering the vertices that highly communicate.

From the valued and locality-labeled graph (VLG) we construct a *reduced valued and locality-labeled graph (RVLG)*. The first step consists into grouping the vertices that highly communicate inside clusters using the *Strongly Connected Components (SCC)* algorithm, leading to reduce the size of the graph as well as the communication between components. Indeed, reducing the graph size permits to decrease the resolution overhead and the network communication. Next, we compute the weights for each cluster and connect them together to create the reduced graph RVLG. To this aim we used the procedures *Vertices Modelization*, and *Edges Connections*.

a. Create the Clusters. The SCC algorithm constructs a cluster from a given vertex a. This cluster will at least contain the element a. To construct the cluster we proceed as follows. First, we initialize all the components of the graph to *non visited* (lines (3, 4) and (13, 14)). Next, we create two sets of

vertices (lines 6, 16). One of them will contain all the successors[4] of all the vertices in this set (lines 7 to 12), and the other one will regroup all the predecessors[5] of all the elements in this set (lines 17 to 22). Finally, we obtain a cluster which is the intersection between the two sets of vertices (line 23).

```
1:  procedure SCC(G(V,E), a in V)
2:      begin
3:      for each v in V
4:          visited(v) = false;
5:      end
6:      V1 = {a};
7:      while ((v in V1) && (not visited(v)))
8:          visited(v) = true;
9:          for each ((u = (v,y) in E) && (y not in V1))
10:             V1 = V1 + {y};
11:         end
12:     end
13:     for each v in V
14:         visited(v) = false;
15:     end
16:     V2 = {a};
17:     while ((v in V2) && (not visited(v)))
18:         visited(v) = true;
19:         for each ((u = (y,v) in E) && (y not in V2))
20:             V2 = V2 + {y};
21:         end
22:     end
23:     return V1 − (V − V2,); // (V1 ⋂ V2)
24:     end
```

b. Create the new Graph. Once the clusters have been created (using the SCC algorithm), we create the *reduced valued and locality-labeled graph (RVLG)*, reduced graph for short[6]. *First*, we define the vertices that compose the reduced graph using the procedure *Vertices Modelization*. Each vertex in the reduced graph corresponds to one of the created clusters (with at least one vertex inside). The cardinality[7] of the reduced graph is equal to the number of created clusters. In what follow, we use the term *cluster* to represent a vertex in the RVLG, in order to distinguish the appellation with the vertices of the VLG. *Next*, we assign weights (time and energy) to each cluster in the reduced graph. The energy and time weights of each cluster c_i are the cumulative weights of all the vertices inside this cluster c_i (lines 8 and 9). *After*, we put a locality label for each cluster in the reduced graph, which depends on the locality labels assigned to the vertices inside these clusters. If at least one vertex j in the original graph (VLG), that is inside a cluster c_i, has a label $l_j = 0$, then the cluster c_i should be labeled with $l_{c_i} = 0$,

[4] A vertex coming after the current vertex in a path (e.g., $G(V,E)$ is a graph, $(x,y) \in E$; y is a successor of x).
[5] A vertex coming before the current vertex in a path (e.g., $G(V,E)$ is a graph, $(x,y) \in E$; x is a predecessor of y).
[6] We use the term RVLG and reduced graph interchangeably.
[7] Number of elements in a set.

otherwise $l_{c_i} = 1$. In other words, if a class from the cluster c_i cannot be offloaded, then the whole cluster c_i cannot be offloaded (lines 11 to 14).

Afterward, we connect the clusters using the *Edges Connections* procedure. For each edge in the original graph (VLG), if the two vertices that constitute the edge belong to two distinct clusters, then an edge between the two clusters (if it does not exist, line 7) is created. *Finally*, we compute the edge weights for the reduced graph. The weights of an edge (c_i, c_j) are equal to the sum of weights of each edge (x, y) in the VLG wherein, x, y do not belong to the same cluster c_i or c_j (i.e., $\left(x \in c_i \; and \; y \in c_j\right)$ or $\left(x \in c_j \; and \; y \in c_i\right)$ (line 8)). This process is repeated for all the clusters. The result is a reduced valued and locality-labeled graph (RVLG).

```
1:  procedure VERTICES_MODELIZATION(G(V,E))
2:      begin
3:      G'(V',E') = G(V,E);
4:      i = 1;
5:      while (V' ≠ ∅)
6:          v = random(V');
7:          C_i = SCC(G'(V',E'), v);
8:          T_{C_i} = Σ_{j=1}^{card(C_i)} T_{v_j}, v_j ∈ C_i;
9:          E_{C_i} = Σ_{j=1}^{card(C_i)} E_{v_j}, v_j ∈ C_i;
10:         V' = V' − C_i;
11:         if ((∃ v_j ∈ C_i) & (l_j == 0)) then
12:             l_{c_i} = 0;
13:         else
14:             l_{c_i} = 1;
15:         i++;
16:     end
17:     return C = ⋃_{j=1}^{i} C_i;
18:     end
```

```
1:  procedure EDGES_CONNECTIONS (G(V,E), C = ⋃_{j=1}^{i} c_i)
2:      begin
3:      for (i = 1, i ≤ C.length, i++)
4:          j = 1;
5:          while ((j ≤ C.lenght) && (j != i))
6:              for each (((u = (v,w) ∈ E) && (v in c_i)) && (w ∈ c_j))
7:                  Create e(c_i,c_j) in the RVLG if it does not exists ;
8:                  W(c_i,c_j) += W(v,w);
9:              end
10:             j++;
11:         end
12:     end
13:     end
```

2. Partitioning. Regarding the decision algorithm (i.e., offload a cluster or not), we propose two different models; *Global-View* and *Local-View*. In the Global-View model, we focus on the optimal solution. To this end we model the offloading decision problem using an ILP. However, the resolution of the ILP may take time and drastically increase with the size of the application. Indeed, there is a local

latency (overhead) of the model resolution to consider in the response time. In the Local-View model, we try to minimize this overhead by proposing a more simple but efficient heuristic solution. Both algorithms take as inputs the reduced version of the graph, the wireless network characteristics (bandwidth and RTT) and the available resources on the UE (energy and computing resources), and return as output the local and remote partitions. For the convenience of presentation, we use the parameters listed in Table A.1 to describe the two models hereafter.

Table 6.1: **List of used parameters**

Variable	Description
T_i	Time Execution of the i^{th} cluster on the UE
E_{cpu}	Energy consumption on the UE to compte the local partition
E_{nic}	Energy consumption by the UE for the uplink and downlink results
E_{idle}	Energy consumption by the UE while the server computes the remote partition
E_{th}	Energy threashold consumption by the UE to execute the application
s	Server speed
UL	Uplink Bandwidth
DL	Downlink bandwidth
d_i	Uplink data (inputs, code) for the offloaded cluster c_i
r_i	Downlink results from the offloaded cluster c_i
RTT	Round trip time (network latency)
l_i	Locality constraint on the cluster c_i

a. Global-View Model. We consider that the total execution time of the application candidate for computation offloading is split into three parts. The first part is the local execution time related to the execution of the local partition of clusters on the UE. The second part is the remote execution time, which considers the execution cost of the remaining clusters (i.e. the remote partition) on the server. Finally, the communication cost, which includes the latency as well as the time to send and receive the results from the server. We define a threshold value, E_{th}, for the energy, as the maximum amount of energy that can be spent for the application processing. We set this value to 95% percent of the total consumption of the application when it is executed locally. This constraint is useful in our model: In the worst case, we gain 5% energy, and more importantly, this constraint prevents from wasting additional energy due to the network communication, as our objective function focuses on the execution time optimization. The algorithm steps are as follows:

First, the RVLG is represented by a matrix in order to be given as input to an ILP. The proposed ILP 6.3 aims to improve the execution cost of the overall program, while maintaining the energy consumption under a certain threshold (E_{th}).

$$Min \sum_{i=1}^{n} \left((1 - x_i) T_i + x_i \left(\frac{T_i}{s} + \frac{d_i}{UL} + \frac{r_i}{DL} + 2 \times RTT \right) \right)$$

$$s.t. \begin{cases} \sum_{i=1}^{n} \left((1 - x_i) E_{cpu} + x_i (E_{idle} + E_{nic}) \right) \leq E_{th} \\ \forall i \in \{1, ..., n\}, \; x_i \in \{0, 1\}, \; l_{c_i} \in \{0, 1\} \\ x_i \leq l_{c_i} \end{cases} \tag{6.3}$$

The symbol x_i is a binary variable that indicates the ILP output: $x_i = 0$ (respectively $x_i = 1$) denotes local (respectively remote) execution of cluster c_i. The constraint $x_i \leq l_{c_i}$ concerns the clusters that cannot be offloaded, due to the hardware dependency constraint, as stated before; therefore, if a cluster is labeled with $l_{c_i} = 0$ (i.e, non-offloaded cluster), then $x_i = 0$ (i.e., this cluster will not be offloaded). The server CPU speed s, could be obtained along with the IP address of the remote server.

Similar to performance, ILP 6.4 aims to minimize the energy consumption of the application through offloading under some constraints; the total execution time of the application should be less or equal to the total execution time on the UE, the symbols x_i as well as l_{c_i} are binary variables, and clusters that have the dependency constraint cannot be offloaded.

$$Min \sum_{i=1}^{n} \left((1 - x_i) E_{cpu} + x_i (E_{idle} + E_{nic}) \right)$$

$$\begin{cases} \sum_{i=1}^{n} \left((1 - x_i) T_i + x_i \left(\frac{T_i}{s} + \frac{d_i}{UL} + \frac{r_i}{DL} + 2RTT \right) \right) \leq \sum_{i=1}^{n} T_i \\ \forall i \in \{1, ..., n\}, \; x_i \in \{0, 1\}, \; l_{c_i} \in \{0, 1\} \\ x_i \leq l_{c_i} \end{cases} \tag{6.4}$$

Next, When the ILP is defined, it is solved to compute the value of each x_i. *Afterward*, each cluster c_i is assigned to one of the two partitions (local partition or remote partition) according to the value of the corresponding x_i; that is, if $x_i = 0$, then the corresponding cluster is assigned to the local partition, otherwise it is enclosed inside the remote one. *Last*, the algorithm returns the set of clusters that should be offloaded (remote partition), and those that should be executed on the UE (local partition).

Algorithm 2 Global-view

Inputs: $M_{n,n}$, $L = \bigcup_{i=1}^{k} (l_{c_i} = 0)$, $k < n$;
Outputs: $X = \bigcup X_{Local}, X_{Cloud}$;
$Objective = Time^0$ *OR* $Energy^1$;
if $(Objective == 0)$ **then**
    $X = \bigcup_{i=1}^{n-k} x_i =$ Solve_ILP $(eq: 6.3)$;
    return X;
else
    if $(Objective == 1)$ **then**
        $X = \bigcup_{i=1}^{n-k} x_i =$ Solve_ILP $(eq: 6.4)$;
        return X;

b. Local-View Model. In the Local-View model, each cluster composing the application is treated separately from the others. The proposed algorithm dissects the possibility of offloading a cluster or not, with respect to the dependency constraint. The purpose is to avoid using the ILP, in order to reduce the resolution time compared with the Global-View algorithm. Indeed, for some low-latency applications, there is a need to decide rapidly to offload a cluster or not, to avoid additional delays. Like the Global-View model, the proposed algorithm accepts as input the RVLG and generates two partitions as output; the local partition and the remote partition. The concept of this algorithm is simple; for each cluster c_i, if either the latency or the cost of communication (i.e., exchanged data) with the ME is higher than the local execution time of this cluster, then offloading the cluster will neither improve the computation cost nor the consumption cost (i.e., $x_i = 0$). Otherwise, if both latency and the communication cost are less than the execution time, the algorithm checks if a gain is achieved when offloading the cluster. The *offloading gain* is the difference between the local cost and the offload cost (the offload cost includes the communication cost and the remote execution) and is defined as:

$$TG_{off} = w \times \left(\frac{1}{s_m} - \frac{1}{s}\right) - \left(\frac{d_i}{UL} + \frac{r_i}{DL} + 2 \times RTT\right) \tag{6.5}$$

where w is the the amount of computation for cluster c_i, and s_m represents the UE processing speed. If offloading a cluster will achieve a gain and no dependency constraint was made on this cluster (i.e., $l_{c_i} = 1$), then it will be offloaded (i.e., $x_i = 1$).

Algorithm 3 Local-view

```
1:  Inputs: M_{n,n}, L = ⋃_{i=1}^{k} (l_{c_i} = 0), k < n;
2:  Outputs: X = ⋃ X_Local, X_Cloud ;
3:  for each i ∈ V
4:      if (i ∈ L) then
5:          x_i = 0
6:      else
7:          if ((RTT < T_i) & (d_i/B_s + r_i/B_r < T_i)) then
8:              if (TG^i_off > 0) then
9:                  x_i = 1
10:             else
11:                 x_i = 0
12:         else
13:             x_i = 0
14: end
15: return X
```

6.3.4 Server Side

We introduce in this section, the architecture and the functionalities of our framework installed on the server. The framework is composed of:

- **Profiling -** Keeps track of several attributes evaluation (e.g., memory usage, CPU time, and energy consumption) during the application execution. *Valgrind*[8], *IntelVTune*[9], and *Visual Studio*[10] are some profilers.
- **Scheduling -** Schedules the execution on the server according to the component dependency graph, generated by the profiler or sent from the UE.
- **S-Proxy -** inserts RPC communication sockets between the UE and the server.
- **Monitoring -** Monitors resources usage. The values are sent to the UE to be used in the offloading process.

Figure 6.11 depicts the architecture of the server framework.

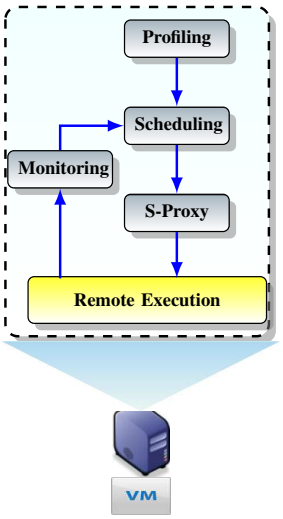


Figure 6.11: **Server framework**

6.4 Results

In this section, we present the evaluation of the proposed framework, through a computer simulation for the MEACOF and a testbed for computation offloading. We describe the simulation scenario and testbed, then discuss the obtained results.

6.4.1 Latency approximation

Our use case is made on the network simulator *NS3*[11] using *lena*[12] project, to simulate an LTE environment wherein, we connect between one to twenty UE with the same budgets and distance to an

[8] http://valgrind.org/
[9] https://software.intel.com/en-us/intel-vtune-amplifier-xe
[10] https://profiler-and-tracer.com/
[11] https://www.nsnam.org/
[12] http://networks.cttc.es/mobile-networks/software-tools/lena/

eNodeB using four bandwidth capacities namely; 1.4 MHz, 5 MHz, 10 MHz, and 20 MHz, then we simulate a communication between these UEs and the eNodeB simultaneously, using UDP packets with 1024 $bytes$ each one.

Table 6.2 lists all Lena network parameters we used in the simulation.

Table 6.2: **Communication simulation parameters**

Parameter	Value
eNB TxPower	30 dBm
N^o of eNBs	1
N^o of UEs	1 to 20
Frequency	1.4 MHz, 5 MHz, 10 MHz, 20 MHz
eNB Antenna Gain	18 dBm
UE Antenna Gain	0
eNB Noise Figure	5 dB
UE Noise Figure	9 dB
Resource Blocks per Cell	25
LTE Scheduler	Round Robin
Duplex Mode	Frequency Division Duplex
Maximum Sending Power	10000 mW
Signal Attenuation Threshold	-110 dBm
Propagation Model	Free Space Model
UE TxPower	26 mW
Path Loss Scenario	URBAN MACROCELL
Packet Size	1024 B
Simulation Time	60 s
Distance UEs to eNB	50 m
MTU	1500 B
LTE RLC Max Buffer Size	1,000,000 × 1024

6.4.2 Offloading Performance

To evaluate the offloading performance of the proposed framework, we developed a testbed composed of a MEC server and a UE performing computation offloading of a face recognition application application (OpenBr[13]). The MEC server is Dell M4800 laptop, powered by an Intel Core i7 processor (clocked at 2.8 Ghz) running Ubuntu 14.04, also including an Nvidia Quadro K2100M GPU card with 2 GB memory. The UE is emulated by a VM running Ubuntu 14.04. The MEC server is 8× faster than the UE. The UL bandwidth is around 50 $Mbps$, and the DL bandwidth is 60 $Mbps$.

In this section we discuss the results obtained through the two proposed algorithms: global-view and local-view.

After the profiling step, 96 classes are used by the application. These classes were then grouped into clusters.

[13] http://openbiometrics.org/

Figure 6.12 illustrates the number of clusters to offload regarding the latency between the UE and the MEC server. We make two main observations here.

- *The number of clusters to offload is inversely proportional to the latency.* This remark is applicable for both proposed algorithms (i.e., Global-View and Local-View). When the network latency is increasing, there is less offloading gain, hence less clusters to offload. Basically, the Global-View (Local-View, respectively) algorithm minimizes (maximizes, respectively) the model defined by equation 6.3 (the gains through equation 6.5, respectively). To minimize equation 6.3, the second part of the sum (i.e., $x_i \left[\frac{T_i}{s} + \frac{d_i}{UL} + \frac{r_i}{DL} + 2 \times RTT \right]$) should be equal to zero when the latency (RTT) is high (i.e., $x_i = 0$). For equation 6.5, there is a gain ($TG_{off} > 0$) when the RTT is less than $\left(\frac{1}{2}\right) \times \left[w \left(\frac{1}{s_m} - \frac{1}{s} \right) - \left(\frac{d_i}{UL} + \frac{r_i}{DL} \right) \right]$. Once the latency reaches $110\,ms$, the performances of both algorithms converge; less than 10 clusters are offloaded. The number of offloaded clusters is merely zero when offloading a cluster will not generate gains.
- *The Global-View algorithm identifies more clusters to offload than the Local-View.* Deciding where to execute each cluster subject to latency constraints is very challenging, as it requires a global view of the program's behavior. The solution is optimal only when the decision strategy is globally optimal (i.e., across the entire program) rather than locally optimal (i.e., relative to a single cluster invocation). The Global-View algorithm solves an ILP, which helps to derive the optimal solution. Thus, for each latency value, the ILP finds the maximum number of clusters that minimizes the cost (i.e., the optimal number of clusters to offload). The Local-View algorithm, on the other hand, focuses on each cluster independently of the others, searching only for local solutions. A cluster is offloaded if and only if a gain (i.e., TG_{off}) is obtained. We believe that the difference between the two curves is due to the communication cost of some clusters which do not result in a local gain as the communication cost exceeds the computation one but improve the global gain.

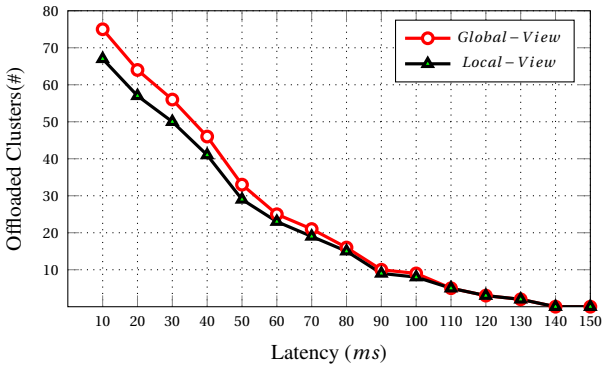


Figure 6.12: **Number of offloaded clusters by report to the RTT between the UE and the MEC sever.**

Now we turn our attention to the gain that could be achieved for users. For this purpose, we focus on two important metrics regarding improving user experience: (i) the time needed to execute the face

recognition application, and (ii) the energy consumed at the user end.
Figure 6.13 presents the response time using three approaches: (i) the whole application is computed locally on the UE, (ii) the application is computed using the Local-View algorithm, and (iii) the application is executed employing the Global-View algorithm. The two last approaches are subject to the network latency. The total execution time for face recognition takes around 9.5 s. This time corresponds to the elapsed time between the moment when the web camera of the laptop is launched and the moment of viewing the result. Inside this delay interval, different actions are performed, including (i) launching the web-camera, (ii) capturing a video (a set of 300 frames), (iii) communication with a graphical QT interface to display the video in real-time, (iv) detection (detecting eyes, face, key-points, and landmarks) for each frame to have the illusion of tracking, (v) normalization (applying color conversion, enhancement, and filtering), (vi) representation (computing binary patterns, key-point descriptors, orientation histograms, and wavelets), (vii) extraction (using clustering, normalization, and quantization), and (viii) matching (using classifiers, distance metrics, and density estimation).

The Global-View algorithm achieves the best performance for all the latency values. For a 10 ms latency, the Global-View algorithm (Local-View, respectively) offers almost 69 (62, respectively) percent of gain compared with the local execution. For small latency (10 ms to 60 ms), the Global-View algorithm is almost 1.13× optimal compared with the Local-View Algorithm. For a latency higher than 110 ms, the two algorithms offer the same performance, since both identify the same clusters to offload. As the latency is high, the two algorithms converge to the same results.

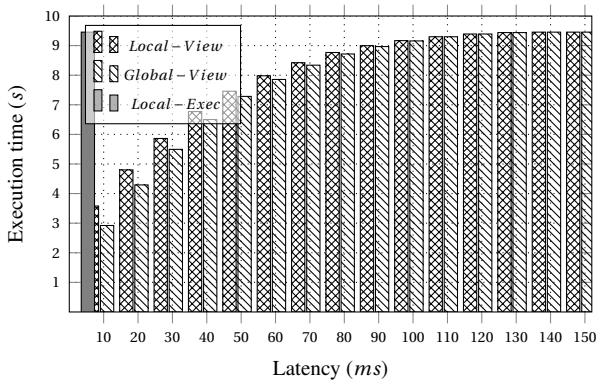


Figure 6.13: **The execution time by report to the RTT between the UE and the MEC sever.**

Figure 6.14 depicts the energy consumed by the face recognition application for the three scenarios (i.e., local execution, Global-View algorithm, and Local-View algorithm). Computing the whole application locally consumes 7.564 J. For 10 ms latency, the Global-View (Local-View, respectively) algorithm offers almost 93.4 (90.5, respectively) percent better performance than the local execution. This gain is inversely proportional to latency. Indeed, when latency increases, the number of clusters to offload decreases, in turn decreasing the achieved gain, which depends on the number of clusters to offload. The Global-View algorithm is almost 3 percent better than the Local-View algorithm. From 140 ms latency,

the achieved gain by the both proposed algorithms is null. Basically, no clusters are offloaded for such high latency. For latencies higher than 140 ms, no gain is achieved by either algorithm, since no clusters are offloaded in such conditions.

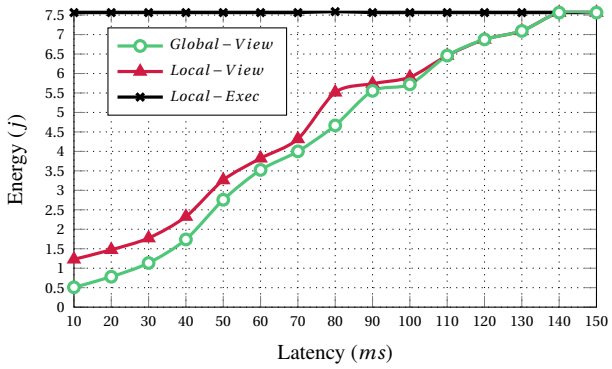


Figure 6.14: **Energy consumption versus the network latency**

6.5 Conclusion

In this chapter, we proposed a novel framework for computation offloading, which uses the MEC architecture defined by the ETSI, taking advantage of its RNIS API to estimate network latency between the UE and a server hosted in the ME. This estimation, coupled with other parameters, is used to derive an offloading decision for the UE. We formulated two models operating at different granularity (application component vs. component cluster level). Whilst the fist model searches for an optimal offloading, the second model searches for a fast solution. The effectiveness of our design, tested using a face recognition application, unveils an execution 62 percent faster than the local execution, and can save up to 93.5 percent of energy for the UE.

CHAPTER 7

CONCLUSION

Nowadays, computation offloading is gaining ground and maturity especially with the transition toward 5G mobile access and edge computing, which solve the issue of high latency with the Cloud. Therefore, offloading delay-sensitive applications becomes possible with these two technologies. 5G mobile network promote the concept of "Anything as a Service" (ANYaaS), which allows MNOs to create and orchestrate 5G services on demand and in a dynamic way. Computation offloading as a Service (COFaaS) is one of these services. Indeed, most of verticals will be connected to the Cloud or edge to offload their requests. In case of automotive, smart cars are interested into collision avoidance and traffic fluidity, therefore, they offload requests to MEC in order to predict and anticipate traffic congestion. For the eHealth, smart clothes, chooses or watches will offload their data/computation to the Cloud/edge to study a consumer health and if needed call the emergency service for intervention. Several other use cases are defined by verticals such as robotics and many others. Yet, Computation offloading can not be deployed before solving the overcoming issues that face this solution raging from resource consumption, interoperability and dependency constraints to security and monetary cost.

We developed through this thesis several algorithms and methods to improve performance of applications via computation offloading, reduce resource consumption and make possible the computation offloading of 3D mobile applications.

7.1 Results obtained during the thesis

The contributions of the thesis were divided into three parts:

- **A computation offloading heuristic for 3D mobile games:** The proposed heuristic first, decomposes the 3D world scene of the game into various game objects including the NPC, player character, environment, and particles. Then enclose for each GO, the requested modules inside

clusters. The heuristic studies the possibility to offload or not a cluster, with respect to the code dependency constraint. It is a cluster decision-making, which focuses on each GO independently from the others. The proposed algorithm accepts as input a cluster and returns a binary decision (i.e., to offload or not the cluster). The heuristic checks if the network latency or the transmission delay is higher than the local execution time of a cluster, then offloading the cluster will not improve the performance, thus it is not offloaded. Otherwise, if both latency and time to exchange data are less than the cluster execution time, the algorithm checks if a gain is obtained by offloading the cluster. The *offloading gain* is the difference between the local cost and the offload cost, this latter includes the communication cost.

- **Two algorithms for computation offloading; *Global-view* and *Local-view*:** The former is based on ILP and the latter is an heuristic. First, the application candidate for computation offloading is modelled with a valued graph, wherein the vertices represent the application components and the edges are the interactions between the components. The vertices are valued with the resource consumption (i.e., CPU time and energy consumption), while the edges are labeled with the amount of exchanged data, frequency of interactions, and code dependency. The graph size is then reduced by clustering the components that highly interact. The global-view algorithm draws an ILP from the reduced graph and solve that ILP with the objective function to reduce the energy consumption and improve performance subject to available energy, execution time, and code dependency. The local-view algorithm computes for each cluster of the reduced graph, the offloading gain. Then decides according to this gain if a cluster will be offloaded or not.
- **A framework-based computation offloading oriented MEC:** The contribution was to devise a framework running on three different entities: (i) *ME application* hosted on ME, which is able to access mobile user related radio information, (ii) *mobile user*, and (iii) *server* hosted in the ME. The ME application is responsible for driving a decision to accept or to reject offloading requests coming from mobile devices, and predicting, using low-level APIs, the RTT value. The mobile device is in charge to take the decision to offload or not the application modules according to the estimated RTT value obtained from the ME application. Finally, the server, hosted on the ME side, computes the offloaded code and sends back the results to the mobile device. As a proof of concept, the authors have run a face recognition application on the mobile device, which offloads computation to the server hosted in the ME.

7.2 Perspectives

To extend our works, the future research directions will cover the following axes: Our frameworks still works in progress, they are unoptimized initial prototypes, with several performance optimizations still possible. In the future we will continue to improve the frameworks to deal with two main issues; the network latency and the rendering activity.

- One of our motivations is to leverage MEC architecture to drive a computation offloading. We want to use MEC as an enabler for low-latency computation offloading-based games. To this aim, we rely on our framework proposed in [188] to orchestrate the offloading process and introduce the concept of *Edge gaming*.
- Another axis that we want to explore is the *caching* concept. Caching allows storing data locally to cope with network latency. Data are used in future invocations to reduce the interaction delay and therefore improve the performance. Data caching approach stores data such as video streams near to the mobile user. This solution is known as CDNs.CDN has been proposed to maximize bandwidth, improve accessibility, and maintain correctness through content replication. We want to investigate service code caching and CDN in our framework to improve the performance.

There are still numerous challenges that need to be solved to make computation offloading a prevalent technique.

- Security and trust issues prohibit the applicability of computation offloading to unknown environments. Introducing SDN controllers and firewalls in computation offloading is a reliable solution. Indeed, an SDN firewall will filter the TCP communications according network security policies by recording and processing the different states of connections and interpreting the possible transitions into an OpenFlow rules.
- Another challenge that current research is facing is mobility of users requesting computation offloading. In the recent trends, users are mobile using their mobile devices such as smartphones to connect to the Internet and consume their applications. Offloading computation in a highly mobile environment is challenging. One of our future motivation is to introduce the SDN/NFV, Follow Me Cloud/Edge [149] to manage the mobility.
- We are also interested into the placement of the offloaded tasks to better exploit the provider resources and improve the performance of the end-user. Offloaded task placement should be similar to the VNF placement. This is central in 5G mobile access, as a tremendous mobile devices are requesting for remote finite resources located in the edge of access network.
- Last but not least, we would like to explore computation offloading from business view. Mobile Network Operators may provide computation offloading as a service in 5G networks. Therefore, we will need to define a model for computation offloading to deal with the 5G architecture and propose a cost model for that service.

APPENDIX A

APPENDIX

A.1 Offloading Gain study

For the convenience of presentation, we used various symbols to describe the following models. Table A.1 lists these symbols and their signification.

Table A.1: **Description of the used symbols**

Variable	Description
w	Amount of computation for P_2
s_m	Mobile speed
s_s	Server speed
B_t	Network transmission bandwidth
B_r	Network reception bandwidth
D_t	Input data and code size to offload
D_r	Results size
RTT	Round trip time (network latency)
P_m	Power consumption on the terminal
P_{comm}	Power consumption for network communication
P_{idle}	Power consumption on the terminal when the server is computing the offloaded tasks
$Avail_{mem}$	The memory size available on the mobile

A.1.1 Improve performance

In this section, we propose an analytic performance-based model for computation offloading. We are interested on identifying when computation offloading brings a gain and what are the parameters impacting this gain. In this model, we suppose that both mobile device and remote server are computing

tasks *sequentially*. We suppose an application requesting for computation offloading. The program of this application is divided into two partitions; the local partition and the remote partition;

- The local partition (P_1) contains the part of the program that should be computed on the mobile device.
- The remote partition (P_2) encloses the part that *may* be offloaded (i.e., the program remainder).

The local partition, P_1, should be computed on the mobile device, it cannot be offloaded according to Section 3.4.4. Therefore, no offloading gain can be achieved through this partition. Hence, we turn our interest on partition P_2.

1. *Local execution of the partition* P_2. The time to execute the partition P_2 on the mobile device is given by: $T_m = \frac{w}{s_m}$
2. *Remote execution of the partition* P_2. The time to execute the partition P_2 on the remote server includes the time to compute the code on the server $\left(\frac{w}{s_s}\right)$ and the communication delay $\left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right)$. It is given by: $T_{off} = \frac{w}{s_s} + \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right)$

Computation offloading improve the application performance when the execution time of the partition P_2 on the remote server is *shorter* than the execution time of this partition P_2 on the mobile device. That is to say; $T_{off} < T_m$. This introduce the Equation A.1

$$\frac{w}{s_s} + \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right) < \frac{w}{s_m} \quad \Rightarrow \frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT < w \times \left(\frac{1}{s_m} - \frac{1}{s_s}\right) \tag{A.1}$$

We note from Equation A.1 that if the execution time of partition P_2 on the mobile device is *shorter* than the communication delay (i.e., $\frac{w}{s_m} < \frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT$) then, even if the server is *infinitely fast*, Equation A.1 will be false (i.e., offloading will not improve the execution time). Hence, this inequality is correct for CPU-computation intensive tasks (Large w), powerful server (high speed s_s), large uplink bandwidth B_t, and downlink bandwidth B_r, and exchanging small data size between the mobile device and the remote server (small d_i).
We define G^t_{off} (Equation A.2) as the *computation offloading gain*.

$$\boldsymbol{G^t_{off} = w \times \left(\frac{1}{s_m} - \frac{1}{s_s}\right) - \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right)} \tag{A.2}$$

A.1.2 Saving energy

Energy consumption is a main constraint for mobile devices, as it cannot be replenished without external source of energy. Several research works have been proposed to study the energy consumption when

performing computation offloading. In what follows, we propose a general model analysis of the energy-related offloading consumption. In this description model, we use the example off the application composed on the local and remote partitions P_1, and P_2 respectively, as described in the section A. Similar to the previous case, computation offloading may improve performance only through remote execution of P_2. Therefore, we focus on this partition.

1. *Local execution of* P_2. The energy consumption is given by: $E_m = P_m \times \frac{w}{s_m}$
2. *Remote execution of* P_2. The energy consumption is given by: $E_{off} = P_{comm} \times \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right) + P_{idle} \times \frac{w}{s_s}$

Computation offloading saves the energy consumption on the mobile device when the energy consumption (on the mobile device) of the partition P_2, when computed remotely is *shorter* than the energy consumption (on the mobile device) of the same partition P_2, when computed locally. Mathematically, $\left(E_{off} < E_m\right)$. This induces Equation A.3

$$P_m \times \frac{w}{s_m} > P_{comm} \times \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right) + P_{idle} \times \frac{w}{s_s} \Rightarrow w \times \left(\frac{P_m}{s_m} - \frac{P_{idle}}{s_s}\right) > P_{comm} \times \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right) \quad \text{(A.3)}$$

This equation holds when: w is large, the remote server is powerful (high speed s_s), the uplink (B_t) and downlink (B_r) bandwidth are large and the data size exchanged is small (small d_i).

The computation offloading gain G^e_{off} is given in Equation A.4

$$\boldsymbol{G^e_{off} = w \times \left(\frac{P_m}{s_m} - \frac{P_{idle}}{s_s}\right) - P_{comm} \times \left(\frac{D_t}{B_t} + \frac{D_r}{B_r} + RTT\right)} \quad \text{(A.4)}$$

A.1.3 Reducing memory

Running some applications with memory intensive on resource-constrained mobile devices such as Personal Digital Assistant (PDA) is not possible. Hopefully, offloading the memory-intensive tasks to a remote server make the execution of these applications on such devices (i.e., PDA) possible. We propose in the following, a simple model analysis of a memory-related offloading consumption.

The memory consumption of the whole application on the mobile device is the sum of the memory occupied by the part P_1 (mem_1) and the part P_2 (mem_2) i.e., ($mem_1 + mem_2$).

Offloading the part P_2 saves the memory utilization when:

$$\begin{cases} mem_1 + mem_2 > Avail_{mem} \times f \\ mem_1 < Avail_{mem} \times f \end{cases} \quad \text{(A.5)}$$

The memory offloading gain is given by Equation A.6.

$$\boldsymbol{G^m_{off} = mem_2 - \delta} \quad \text{(A.6)}$$

Where δ includes the size of the results sent from the remote server to the mobile device and the size of the offloading framework.

To sum up, we proposed here above, three simple mathematical models to study how to fulfill the requirements of applications and how to calculate the computation offloading gain. However, the number of parameters involved in the optimization models varies between frameworks.

A.2 Games

A.2.1 Rendering Pipelines

Unity 3D uses four additional rendering activities, which occur in conjunction with the main pipeline. We summarize these rendering activities hereafter.

1. *Forward rendering* – It has been introduced for dynamic lighting in 3D environments. It consists of three passes: **(i)** *Ambient Pass* applies a global low lighting to the overall 3D scene. The results of this pass are saved in the frame buffer; **(ii)** *Light Pass* repeats the drawing process for each opaque object, affected by one of the light sources regardless of others in the scene. The lighting is accumulated in the frame buffer; **(iii)** *Transparency Pass* draws all the transparent objects in the same way as for opaque objects, but with a drawing order of transparent objects from back to front. The transparent geometry is added to the frame buffer and combined with the ambient and opaque results [62].
2. *Deferred shading* – The idea is to defer the lighting calculation to the second pass until all the geometry has been rendered. The deferred shader algorithm uses smart management of different buffers [244], defined through three passes as follows: **(i)** *Geometry Pass* (Opaque Pass) applies an ambient lighting, saves the results in the frame buffer, and fills the Geometry buffer; **(ii)** *Light Pass* processes the lighting through the buffers transition calculations, starting with a *depth buffer* to rebuild origin position of pixels. The results are added to a *normal buffer* data to calculate the diffuse light. Then, a specular lighting is calculated using the *position*, the *normal*, and *specular data buffers*. Finally, the specular light is applied to *colour* and *shading data buffer* and accumulated with all the other light sources. The global colour of these buffers is accumulated in the frame buffer; **(iii)** *Transparency Pass* renders the transparent geometry using the forward rendering (Ambient Pass and Light Pass) [62, 252].
3. *Pre-pass rendering* – Similarly as deferred shading, it addresses the restricted usage of different material shaders. The lighting is stored in the new buffer with light pre-pass rendering, instead of applying it to the colour and shading data buffer. This process follows four passes: **(i)** *Geometry Pass* applies the ambient lighting, draws the opaque geometry, and fills the G-buffer; **(ii)** *Light Pass* performs the lighting calculations in the pixel shader using the G-buffer for all light sources. The results are accumulated with additive primitive in lighting buffer; **(iii)** *Material Pass* draws

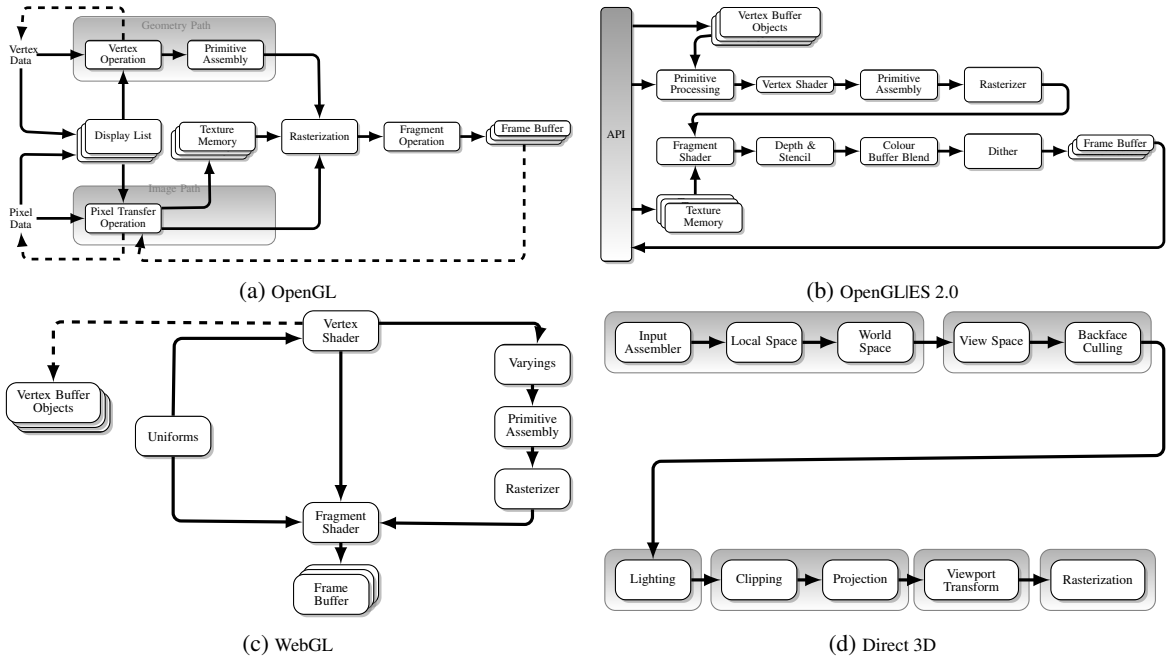


Figure A.1: **Graphic pipelines stages**

again the geometry using the lighting buffer as lighting input for the material-specific shader; **(iv)** *Transparency Pass* uses the forward rendering (Ambient Pass & Light Pass) to process the lighting. The results are saved in the frame buffer.

4. *Vertex lit* – This operation is the fastest one and is supported by a large number of hardware. This process is done in one pass, wherein each object is rendered with lighting calculated on the vertices of the object from all the light sources.

A.2.2 Game Description

Viking Village – is a FPS 3D-game. The sequence offers a look at a medieval Viking village. The scene is characterized by high design quality with many details and various effects like water vibration, fire and smoke particles, sunlight, firelight, shadows, and reflecting lights, shadows, and colours by surfaces.

Tower Bridge Defense – is a TPS 2D platformer game. It depicts a player character fighting against NPC in a physics-driven 2D sample level. The avatars jump between suspended platforms, over obstacles, to advance the game. It features many objects subject to the 2D physics such as gravity, velocity, and other forces.

Stealth – is a TPS 3D-game that describes a hostile environment characterized with guards (NPC) and security cameras, laser gates, key-card, and elevator. The game objects have a high rendering quality. The PC and NPC have an advanced animators, and the AI that manages NPC, laser gates, key-cards, and elevator is complex.

Survival Shooter – is a TPS 3D action-game, wherein the gamer fights against an NPC. The objective is to shoot them, eliminate as many as possible, stay alive and try to get a high score. The game is characterized by 3D physics, character animations, and many NPC.

Tanks – is a 2D MMOG where two players fight in a tank shooting game in a hostile desert environment. It is characterized by few graphical add-ons (such as rocks, palm trees, and rocky mountains). The tanks are subject to physics (explosion particles, velocity, and friction), and directional light to simulate the sun.

Space Shooter – is a 2D TPS game that takes place in space. The game is described by a large number of spacial-ships (enemies) and asteroids objects that are subject to physics: explosion particles, velocity and friction and, AI management.

Car – is a 3D racing game, with platforms. It is a standard scene, representing a car with a whole engine system (including speedup, stop, red light, turning left and right, and engine sound). The scene includes some obstacles (static objects) and has platforms like bridges and pings. The terrain is a car road with obstacles and platforms around.

Unity Lab – is a 3D TPS game with a simulation about the daily life of Dr. Charles Francis, a research scientist at Unity Lab. The Unity lab includes different rooms reachable through hallways, with dynamic doors and elevators. The environment has an improved graphics, including shading, cinematic image effects, particles systems, and lighting. The NPC is represented by a flying robot.

Multiplayer FPS – is a 3D MMOG FPS game describing a player fighting against a NPC inside an arena. The player character is a flying robot with blasters. The NPC is a humanoid character. The game is characterized with various effects for the arena such as fires, smokes, lights, sun, and storm.

A.2.3 Game Classification

- (i) Classification per Playability

We start by collecting the data from Figures 4.4, 4.5, and 4.6, which enable to classify the nine games on each platform with the two encoding qualities as *playable* or *non-playable*. Next, we use the system A.7 to compact the classification through Table A.2.

$$\text{(A.7)} \qquad \begin{cases} Playability \wedge \neg(Playability) \Rightarrow \neg(Playability) \\ Playability \wedge Playability \Rightarrow Playability \\ \neg(Playability) \wedge \neg(Playability) \Rightarrow \neg(Playability) \end{cases}$$

- (ii) Classification per Resource variability

We attribute the predicate *high variability* or *low variability* for each game on each platform and for each quality to the results obtained in Figures 4.4, 4.5, and 4.6. Then, we use the system A.8, to compact the classification through Table A.3.

Table A.2: **Games classification per playability**

	Dell M4800	Surface Pro	HTC M8	Galaxy S6 Edge	Browsers
Viking Village	Non-Playable	Non-Playable	Non-Playable	Non-Playable	Non-Playable
Tower Bridge	Playable	Playable	Playable	Playable	Playable
Stealth	Non-Playable	Non-Playable	Non-Playable	Non-Playable	Non-Playable
Survival Shooter	Playable	Playable	Non-Playable	Non-Playable	Playable
Tanks	Playable	Playable	Playable	Playable	Playable
Space Shooter	Playable	Playable	Playable	Playable	Playable
Car	Playable	Playable	Non-Playable	Non-Playable	Playable
Unity Lab	Non-Playable	Non-Playable	Non-Playable	Non-Playable	Non-Playable
Multiplayer	Non-Playable	Non-Playable	Non-Playable	Non-Playable	Non-Playable

$$\text{(A.8)} \quad \begin{cases} HighVariability \wedge HighVariability \Rightarrow HighVariability \\ HighVariability \wedge LowVariability \Rightarrow HighVariability \\ LowVariability \wedge LowVariability \Rightarrow LowVariability \end{cases}$$

Table A.3: **Games classification per variability**

	Dell M4800	Surface Pro	HTC M8	Galaxy S6 Edge	Browsers
Viking Village	High Variability	High Variability	High Variability	High Variability	High Variability
Tower Bridge	Low Variability	Low Variability	Low Variability	Low Variability	Low Variability
Stealth	High Variability	High Variability	High Variability	High Variability	High Variability
Survival Shooter	Low Variability	High Variability	Low Variability	Low Variability	Low Variability
Tanks	Low Variability	Low Variability	Low Variability	Low Variability	Low Variability
Space Shooter	Low Variability	Low Variability	Low Variability	Low Variability	Low Variability
Car	Low Variability	Low Variability	Low Variability	Low Variability	Low Variability
Unity Lab	High Variability	High Variability	High Variability	High Variability	High Variability
Multiplayer	Low Variability	High Variability	High Variability	High Variability	Low Variability

- (iii) Classification per Playability and Resource variability

Joining Table A.2 and Table A.3, by applying the system A.9, we obtain the classification of the different games (*i.e.,* Table IV : "Best option for architecture implementation per game and device").

$$\text{(A.9)} \quad \begin{cases} Playable \Rightarrow Traditional \\ \neg(Playability) \wedge LowVariability \Rightarrow Cloud \\ \neg(Playability) \wedge HighVariability \Rightarrow Offload \end{cases}$$

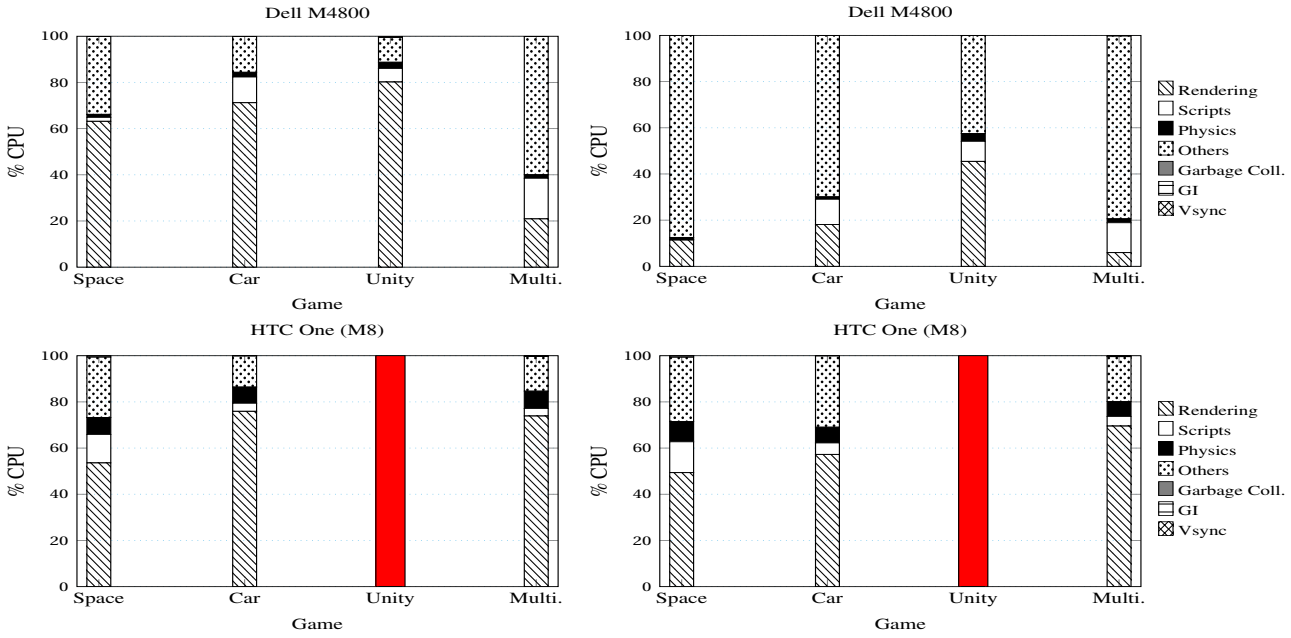
A.2.4 Client and Server Devices

Table A.4 presents the devices used for each architecture to do our experiments. Symbol (†) represents the client, and (*) the server.

Table A.4: **The chosen device for each game under the three architectures**

Games	Traditional Architecture	Cloud Gaming	Computation Offloading
Tanks	Dell Precision M4800	Dell Precision M4800† Dell Tower Machine*	Dell Precision M4800† Dell Tower Machine*
Viking Village	Dell Precision M4800	Dell Precision M4800† Dell Tower Machine*	Dell Precision M4800† Dell Tower Machine*
Stealth	Dell Precision M4800	Dell Precision M4800† Dell Tower Machine*	Dell Precision M4800† Dell Tower Machine*
Space Shooter	HTC One (M8)	HTC One (M8)† Dell Tower Machine*	HTC One (M8)† Dell Tower Machine*
Car	HTC One (M8)	HTC One (M8)† Dell Tower Machine*	HTC One (M8)† Dell Tower Machine*
Unity Lab	Dell Precision M4800	Dell Precision M4800† Dell Tower Machine*	Dell Precision M4800† Dell Tower Machine*
Multiplayer	HTC One (M8)	HTC One (M8)† Dell Tower Machine*	HTC One (M8)† Dell Tower Machine*

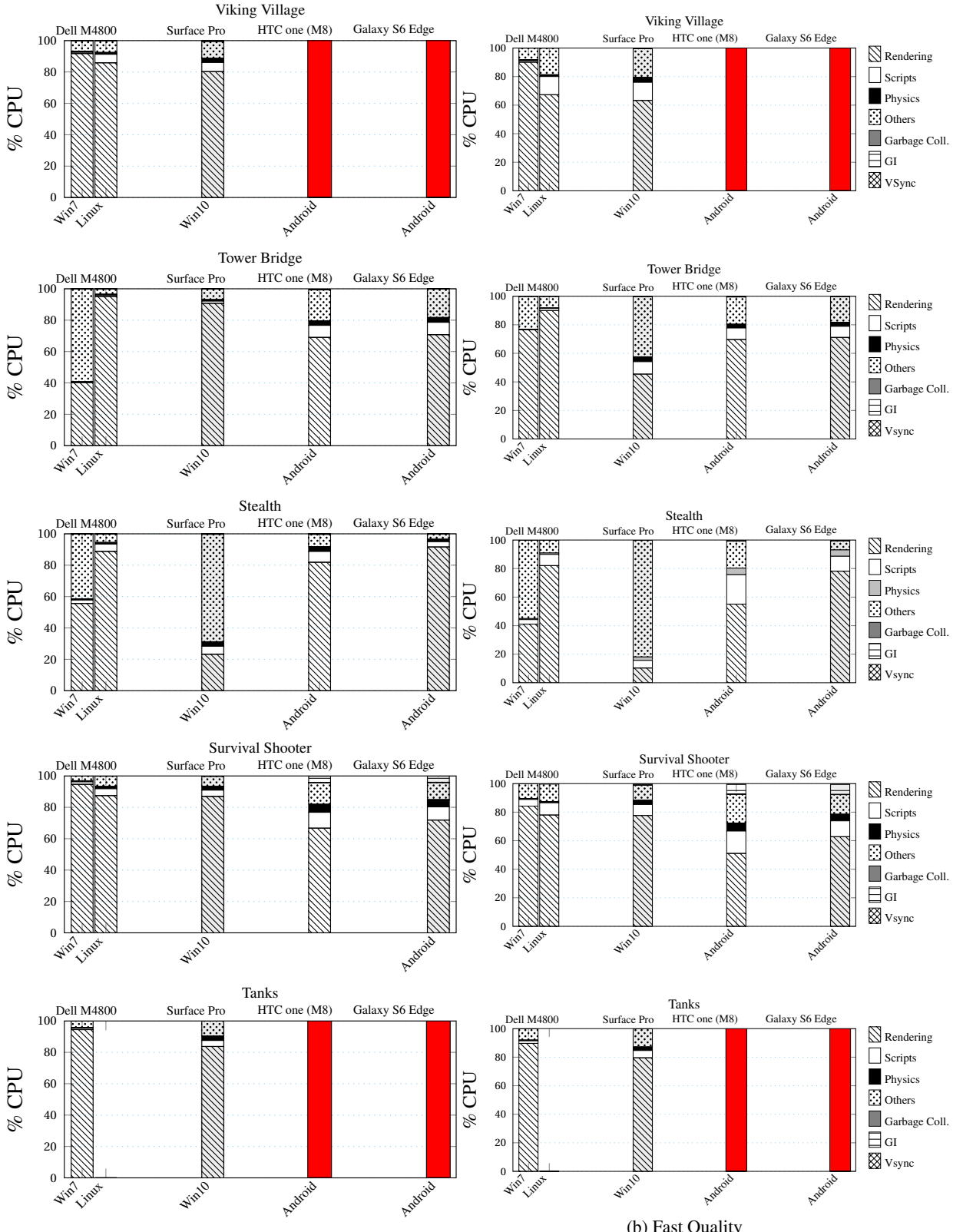
A.2.5 Modular results



(a) Good quality

(b) Fast quality

Figure A.2: **How the CPU time is divided among modules on Dell M4800 and HTC One (M8)**



(a) Good Quality

(b) Fast Quality

Figure A.3: **How the CPU time is divided among modules on different targets**

BIBLIOGRAPHY

[1] *Unity: The leading global game industry software*, accessed in Aug. 2015.

[2] 3GPP, *General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access*, Technical Specification (TS) 23.401, 3rd Generation Partnership Project (3GPP), 12 2017.
Version 15.2.0.

[3] ——, *Policy and charging control architecture*, Technical Specification (TS) 23.203, 3rd Generation Partnership Project (3GPP), 12 2017.
Version 15.1.0.

[4] M. AAZAM AND E. HUH, *Fog computing and smart gateway based communication for cloud of things*, in 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, Barcelona, Spain, August 27-29, 2014, 2014, pp. 464–470.

[5] E. ABEBE AND C. RYAN, *A hybrid granularity graph for improving adaptive application partitioning efficacy in mobile computing environments*, in Proceedings of The Tenth IEEE International Symposium on Networking Computing and Applications, NCA 2011, August 25-27, 2011, Cambridge, Massachusetts, USA, 2011, pp. 59–66.

[6] ——, *Adaptive application offloading using distributed abstract class graphs in mobile environments*, Journal of Systems and Software, 85 (2012), pp. 2755–2769.

[7] S. ABOLFAZLI, Z. SANAEI, AND E. AHMED, *Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges*, IEEE Communications Surveys and Tutorials, 16 (2014), pp. 337–368.

[8] S. ABOLFAZLI, Z. SANAEI, A. GANI, AND M. SHIRAZ, *MOMCC: market-oriented architecture for mobile~cloud~computing based on service~oriented~architecture*, CoRR, abs/1206.6209 (2012).

[9] G. ANANTHANARAYANAN, C. DOUGLAS, R. RAMAKRISHNAN, S. RAO, AND I. STOICA, *True elasticity in multi-tenant data-intensive compute clusters*, in ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012, 2012, p. 24.

[10] E. F. ANDERSON, S. ENGEL, P. COMNINOS, AND L. MCLOUGHLIN, *The case for research in game engine architecture*, in Proc of the 2008 Conf. on Future Play, 2008.

[11] P. ANGIN AND B. K. BHARGAVA, *An agent-based optimization framework for mobile-cloud computing*, JoWUA, 4 (2013), pp. 1–17.

[12] G. J. ARMITAGE AND A. HEYDE, *REED: optimizing first person shooter game server discovery using network coordinates*, TOMCCAP, 8 (2012), p. 20.

[13] K. ARNOLD, R. SCHEIFLER, J. WALDO, B. O'SULLIVAN, AND A. WOLLRATH, *Jini specification*, Addison-Wesley Longman Publishing Co., Inc., 1999.

[14] L. BADGER, T. GRANCE, R. PATT-CORNER, AND J. VOAS, *Draft cloud computing synopsis and recommendations*, NIST special publication, 800 (2011), p. 146.

[15] A. BAID, R. MADAN, AND A. SAMPATH, *Delay estimation and fast iterative scheduling policies for LTE uplink*, in 10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), Paderborn, Germany, May 14-18, 2012, 2012, pp. 89–96.

[16] P. BALAKRISHNAN AND C.-K. THAM, *Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing*, in Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, IEEE Computer Society, 2013, pp. 34–41.

[17] R. K. BALAN, D. GERGLE, M. SATYANARAYANAN, AND J. D. HERBSLEB, *Simplifying cyber foraging for mobile devices*, in Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007), San Juan, Puerto Rico, June 11-13, 2007, 2007, pp. 272–285.

[18] R. K. BALAN, M. SATYANARAYANAN, S. PARK, AND T. OKOSHI, *Tactics-based remote execution for mobile computing*, in Proceedings of the First International Conference on Mobile Systems, Applications, and Services, MobiSys 2003, San Francisco, CA, USA, May 5-8, 2003, 2003.

[19] T. BALL AND J. R. LARUS, *Optimally profiling and tracing programs*, ACM Trans. Program. Lang. Syst., 16 (1994), pp. 1319–1360.

[20] R. BALLAGAS, S. G. KRATZ, J. O. BORCHERS, E. YU, S. P. WALZ, C. O. FUHR, L. HOVESTADT, AND M. TANN, *Rexplorer: a mobile, pervasive spell-casting game for tourists*, in Extended Abstracts Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007, 2007, pp. 1929–1934.

[21] Y. M. BEGUM AND M. A. M. MOHAMED, *A dht-based process migration policy for mobile clusters*, in Seventh International Conference on Information Technology: New Generations, ITNG 2010, Las Vegas, Nevada, USA, 12-14 April 2010, 2010, pp. 934–938.

[22] P. B. BESKOW, A. PETLUND, AND G. A. ERIKSTAD, *Reducing Game Latency by Migration, Core-selection and TCP Modifications*, International Journal of Advanced Media and Communication, 4 (2010), pp. 343–363.

[23] E. BIALIC, *Wysips® connect, the first solution for the indoor/outdoor vlc lighting saturation problematics*, (2015).

[24] G. S. BLAIR AND P. GRACE, *Emergent middleware: Tackling the interoperability problem*, IEEE Internet Computing, 16 (2012), pp. 78–82.

[25] J. BOLOT, *Characterizing end-to-end packet delay and loss in the internet*, J. High Speed Networks, 2 (1993), pp. 305–323.

[26] K. D. BOWERS, A. JUELS, AND A. OPREA, *HAIL: a high-availability and integrity layer for cloud storage*, in Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009, 2009, pp. 187–198.

[27] W. BROOKER, *Hunting the Dark Knight: Twenty-First Century Batman*, 2012.

[28] T. D. BURD AND R. W. BRODERSEN, *Design issues for dynamic voltage scaling*, in Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000, Rapallo, Italy, July 25-27, 2000, 2000, pp. 9–14.

[29] W. BÜSCHEL, P. REIPSCHLÄGER, AND R. DACHSELT, *Foldable3d: Interacting with 3d content using dual-display devices*, in Proceedings of the 2016 ACM on Interactive Surfaces and Spaces, ISS 2016, Niagara Falls, ON, Canada, November 6-9, 2016, 2016, pp. 367–372.

[30] E. BUYUKKAYA, M. ABDALLAH, AND G. SIMON, *A survey of peer-to-peer overlay approaches for networked virtual environments*, Peer-to-Peer Networking and Applications, 8 (2015), pp. 276–300.

[31] R. BUYYA, C. S. YEO, S. VENUGOPAL, J. BROBERG, AND I. BRANDIC, *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*, Future Generation Comp. Syst., 25 (2009), pp. 599–616.

[32] G. H. CÁNEPA AND D. LEE, *An adaptable application offloading scheme based on application behavior*, in 22nd International Conference on Advanced Information Networking and Applications, AINA 2008, Workshops Proceedings, GinoWan, Okinawa, Japan, March 25-28, 2008, 2008, pp. 387–392.

[33] M. CANO AND G. DOMÉNECH-ASENSI, *A secure energy-efficient m-banking application for mobile devices*, Journal of Systems and Software, 84 (2011), pp. 1899–1909.

[34] V. CARDELLINI, V. D. N. PERSONE, V. D. VALERIO, F. FACCHINEI, V. GRASSI, F. L. PRESTI, AND V. PICCIALLI, *A game-theoretic approach to computation offloading in mobile cloud computing*, Math. Program., 157 (2016), pp. 421–449.

[35] C.-Y. CHANG, A. KONSTANTINOS, N. NAVID, AND S. THRASYVOULOS, *Analyzing mec architectural implications for lte/lte-a*, tech. rep., Tech. rep., Eurecom, 2016.

[36] G. CHEN, B. KANG, M. T. KANDEMIR, N. VIJAYKRISHNAN, M. J. IRWIN, AND R. CHANDRAMOULI, *Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices*, IEEE Trans. Parallel Distrib. Syst., 15 (2004), pp. 795–809.

[37] H.-Y. CHEN, Y.-H. LIN, AND C.-M. CHENG, *Coca: Computation offload to clouds using aop*, in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, IEEE, 2012, pp. 466–473.

[38] M. CHEN, Y. HAO, Y. LI, C. LAI, AND D. WU, *On the computation offloading at ad hoc cloudlet: architecture and service modes*, IEEE Communications Magazine, 53 (2015), pp. 18–24.

[39] M. CHEN, Y. HAO, M. QIU, J. SONG, D. WU, AND I. HUMAR, *Mobility-aware caching and computation offloading in 5g ultra-dense cellular networks*, Sensors, 16 (2016), p. 974.

[40] N. CHEN, Y. CHEN, Y. YOU, H. LING, P. LIANG, AND R. ZIMMERMANN, *Dynamic urban surveillance video stream processing using fog computing*, in IEEE Second International Conference on Multimedia Big Data, BigMM 2016, Taipei, Taiwan, April 20-22, 2016, 2016, pp. 105–112.

[41] S. CHEN, Y. WANG, AND M. PEDRAM, *A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud*, in 2013 IEEE Global Communications Conference, GLOBECOM 2013, Atlanta, GA, USA, December 9-13, 2013, 2013, pp. 2885–2890.

[42] X. CHEN, *Decentralized computation offloading game for mobile cloud computing*, IEEE Trans. Parallel Distrib. Syst., 26 (2015), pp. 974–983.

[43] Y.-F. CHEN, D. GIBBON, R. JANA, B. RENGER, D. STERN, M. YANG, B. WEI, AND H. SUN, *Project geotv: a three-screen service*, in Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, IEEE Press, 2009, pp. 58–59.

[44] R. CHOW, P. GOLLE, M. JAKOBSSON, E. SHI, J. STADDON, R. MASUOKA, AND J. MOLINA, *Controlling data in the cloud: outsourcing computation without outsourcing control*, in Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009, 2009, pp. 85–90.

[45] S. CHOY, B. WONG, G. SIMON, AND C. ROSENBERG, *The brewing storm in cloud gaming: A measurement study on cloud to end-user latency*, in 11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012, 2012, pp. 1–6.

[46] S. CHOY, B. WONG, G. SIMON, AND C. ROSENBERG, *The brewing storm in cloud gaming: A measurement study on cloud to end-user latency*, in Proceedings of the 11th ACM/IEEE Netgames workshop, 2012.

[47] ——, *A hybrid edge-cloud architecture for reducing on-demand gaming latency*, Multimedia Systems, 20 (2014), pp. 503–519.

[48] H. CHU, H. SONG, C. WONG, S. KURAKAKE, AND M. KATAGIRI, *Roam, a seamless application framework*, Journal of Systems and Software, 69 (2004), pp. 209–226.

[49] S. CHUAH, C. YUEN, AND N. CHEUNG, *Cloud gaming: a green solution to massive multiplayer online games*, IEEE Wireless Commun., 21 (2014), pp. 78–87.

[50] B. CHUN, S. IHM, P. MANIATIS, M. NAIK, AND A. PATTI, *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, 2011, pp. 301–314.

[51] B. CHUN AND P. MANIATIS, *Augmented smartphone applications through clone cloud execution*, in Proceedings of HotOS'09: 12th Workshop on Hot Topics in Operating Systems, May 18-20, 2009, Monte Verità, Switzerland, 2009.

[52] B.-G. CHUN AND P. MANIATIS, *Augmented smartphone applications through clone cloud execution.*, in HotOS, vol. 9, 2009, pp. 8–11.

[53] ——, *Dynamically partitioning applications between weak devices and clouds*, in Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, ACM, 2010, p. 7.

[54] M. CLAYPOOL AND K. CLAYPOOL, *Perspectives, frame rates and resolutions: it's all in the game*, in Proceedings of the 4th ACM International Conference on Foundations of Digital Games, 2009.

[55] M. CLAYPOOL AND K. T. CLAYPOOL, *Latency and player actions in online games*, Commun. ACM, 49 (2006), pp. 40–45.

[56] ——, *Latency can kill: precision and deadline in online games*, in Proceedings of the First Annual ACM MMSys Conference, 2010.

[57] S. CONSORTIUM ET AL., *Salutation architecture specification version 2.1*, Salutation, June, (1999).

[58] B. COWAN AND B. KAPRALOS, *A survey of frameworks and game engines for serious game development*, in Proceedings of the 14th IEEE International Conference on Advanced Learning Technologies (ICALT), 2014.

[59] E. CUERVO, A. BALASUBRAMANIAN, D. CHO, A. WOLMAN, S. SAROIU, R. CHANDRA, AND P. BAHL, *MAUI: making smartphones last longer with code offload*, in Proceedings of

the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010, 2010, pp. 49–62.

[60] E. CUERVO, A. WOLMAN, L. P. COX, K. LEBECK, A. RAZEEN, S. SAROIU, AND M. MUSUVATHI, *Kahawai: High-quality mobile gaming using GPU offload*, in Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2015, Florence, Italy, May 19-22, 2015, 2015, pp. 121–135.

[61] K. DEB, S. AGRAWAL, A. PRATAP, AND T. MEYARIVAN, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Trans. Evolutionary Computation, 6 (2002), pp. 182–197.

[62] M. DEERING, S. WINNER, AND B. SCHEDIWY, *The triangle processor and normal vector shader: a VLSI system for high performance graphics*, in Proc. of 15th Conf. on Comp. Graphics and Interactive Techniques, SIGGRAPH, 1988, pp. 21–30.

[63] S. DENG, L. HUANG, J. TAHERI, AND A. Y. ZOMAYA, *Computation offloading for service workflow in mobile cloud computing*, IEEE Trans. Parallel Distrib. Syst., 26 (2015), pp. 3317–3329.

[64] S. DENG, L. HUANG, J. TAHERI, AND A. Y. ZOMAYA, *Computation offloading for service workflow in mobile cloud computing*, IEEE Transactions on Parallel and Distributed Systems, 26 (2015), pp. 3317–3329.

[65] M. DERUYCK, W. VEREECKEN, E. TANGHE, W. JOSEPH, M. PICKAVET, L. MARTENS, AND P. DEMEESTER, *Comparison of power consumption of mobile wimax, HSPA and LTE access networks*, in 9th Conference on Telecommunications Internet and Media Techno Economics, CTTE 2010, Ghent, Belgium, June 7-9, 2010, 2010, pp. 1–7.

[66] J. T. DOSWELL, *Augmented learning: Context-aware mobile augmented reality architecture for learning*, in Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT 2006, Kerkrade, The Netherlands, July 5-7, 2006, 2006, pp. 1182–1183.

[67] A. J. DOU, V. KALOGERAKI, D. GUNOPULOS, T. MIELIKÄINEN, AND V. H. TUULOS, *Misco: a mapreduce framework for mobile systems*, in Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, PETRA 2010, Samos, Greece, June 23-25, 2010, 2010.

[68] A. B. DOWNEY, *Using pathchar to estimate internet link characteristics*, in SIGCOMM, 1999, pp. 241–250.

[69] D. DURKEE, *Why cloud computing will never be free*, Commun. ACM, 53 (2010), pp. 62–69.

[70] S. ECHEVERRÍA, J. ROOT, B. BRADSHAW, AND G. A. LEWIS, *On-demand VM provisioning for cloudlet-based cyber-foraging in resource-constrained environments*, in 6th International Conference on Mobile Computing, Applications and Services, MobiCASE 2014, Austin, TX, USA, November 6-7, 2014, 2014, pp. 116–124.

[71] H. EOM, R. J. O. FIGUEIREDO, H. CAI, Y. ZHANG, AND G. HUANG, *MALMOS: machine learning-based mobile offloading scheduler with online training*, in 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2015, San Francisco, CA, USA, March 30 - April 3, 2015, 2015, pp. 51–60.

[72] H. EOM, P. S. JUSTE, R. J. O. FIGUEIREDO, O. TICKOO, R. ILLIKKAL, AND R. IYER, *Machine learning-based runtime scheduler for mobile offloading framework*, in IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013, 2013, pp. 17–25.

[73] ETSI, *Mobile edge computing (mec); framework and reference architecture*, ETSI GS MEC, 3 (2016), p. V1.1.1.

[74] ——, *Mobile Edge Computing (MEC); Technical Requirements*, Group Specification (GS) 002, European Telecommunication Standard Institute (ETSI), 03 2016.
Version 1.1.1.

[75] ——, *Mobile Edge Computing (MEC); Terminology*, Group Specification (GS) 001, European Telecommunication Standard Institute (ETSI), 03 2016.
Version 1.1.1.

[76] W.-C. FENG AND W.-C. FENG, *On the geographic distribution of on-line game servers and players*, in Proceedings of the 2nd ACM Netgames workshop, 2003.

[77] L. L. FERREIRA, G. D. SILVA, AND L. M. PINHO, *Service offloading in adaptive real-time systems*, in IEEE 16th Conference on Emerging Technologies & Factory Automation, ETFA 2011, Toulouse, France, September 5-9, 2011, 2011, pp. 1–6.

[78] D. FESEHAYE, Y. GAO, K. NAHRSTEDT, AND G. WANG, *Impact of cloudlets on interactive mobile cloud applications*, in 16th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2012, Beijing, China, September 10-14, 2012, 2012, pp. 123–132.

[79] D. FESEHAYE, Y. GAO, K. NAHRSTEDT, AND G. WANG, *Impact of cloudlets on interactive mobile cloud applications*, in Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International, IEEE, 2012, pp. 123–132.

[80] J. A. FISHER, J. R. ELLIS, J. C. RUTTENBERG, AND A. NICOLAU, *Parallel processing: a smart compiler and a dumb machine (with retrospective)*, in 20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation 1979-1999, A Selection, 1984, pp. 112–124.

[81] J. FLINN, S. PARK, AND M. SATYANARAYANAN, *Balancing performance, energy, and quality in pervasive computing*, in Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, IEEE, 2002, pp. 217–226.

[82] J. FLINN, S. PARK, AND M. SATYANARAYANAN, *Balancing performance, energy, and quality in pervasive computing*, in 22nd International Conferenceon Distributed Computing Systems (ICDCS02), Vienna, Austria, 2002, pp. 217–226.

[83] G. FOLINO AND F. S. PISANI, *A framework for modeling automatic offloading of mobile applications using genetic programming*, in European Conference on the Applications of Evolutionary Computation, Springer, 2013, pp. 62–71.

[84] R. FRIEDMAN AND N. HAUSER, *COARA: code offloading on android with aspectj*, CoRR, abs/1604.00641 (2016).

[85] S. GARRISS, R. CÁCERES, S. BERGER, R. SAILER, L. VAN DOORN, AND X. ZHANG, *Trustworthy and personalized computing on public kiosks*, in Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys 2008), Breckenridge, CO, USA, June 17-20, 2008, 2008, pp. 199–210.

[86] Y. GE, Y. ZHANG, Q. QIU, AND Y. LU, *A game theoretic resource allocation for overall energy minimization in mobile cloud computing system*, in International Symposium on Low Power Electronics and Design, ISLPED'12, Redondo Beach, CA, USA - July 30 - August 01, 2012, 2012, pp. 279–284.

[87] C. GENTRY, *Fully homomorphic encryption using ideal lattices*, in Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, 2009, pp. 169–178.

[88] ——, *Computing arbitrary functions of encrypted data*, Commun. ACM, 53 (2010), pp. 97–105.

[89] I. GIURGIU, O. RIVA, AND G. ALONSO, *Dynamic software deployment from clouds to mobile devices*, in ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2012, pp. 394–414.

[90] I. GIURGIU, O. RIVA, AND G. ALONSO, *Dynamic software deployment from clouds to mobile devices*, in Middleware 2012 - ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings, 2012, pp. 394–414.

[91] I. GIURGIU, O. RIVA, D. JURIC, I. KRIVULEV, AND G. ALONSO, *Calling the cloud: Enabling mobile phones as interfaces to cloud applications*, in Middleware 2009, ACM/IFIP/USENIX, 10th International Middleware Conference, Urbana, IL, USA, November 30 - December 4, 2009. Proceedings, 2009, pp. 83–102.

[92] M. GORACZKO, J. LIU, D. LYMBEROPOULOS, S. MATIC, B. PRIYANTHA, AND F. ZHAO, *Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems*, in Proceedings of the 45th annual design automation conference, ACM, 2008, pp. 191–196.

[93] S. GOYAL AND J. CARTER, *A lightweight secure cyber foraging infrastructure for resource-constrained devices*, in 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004), 2-10 December 2004, Lake District National Park, UK, 2004, pp. 186–195.

[94] X. GU, A. MESSER, I. GREENBERG, D. S. MILOJICIC, AND K. NAHRSTEDT, *Adaptive offloading for pervasive computing*, IEEE Pervasive Computing, 3 (2004), pp. 66–73.

[95] X. GU, K. NAHRSTEDT, A. MESSER, I. GREENBERG, AND D. S. MILOJICIC, *Adaptive offloading inference for delivering applications in pervasive computing environments*, in Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), March 23-26, 2003, Fort Worth, Texas, USA, 2003, pp. 107–114.

[96] T. GUAN, E. ZALUSKA, AND D. D. ROURE, *A grid service infrastructure for mobile devices*, in 2005 International Conference on Semantics, Knowledge and Grid (SKG 2005), 27-29 November 2005, Beijing, China, 2005, p. 42.

[97] S. GUNDAVELLI, K. LEUNG, V. DEVARAPALLI, K. CHOWDHURY, AND B. PATIL, *Proxy mobile ipv6*, tech. rep., 2008.

[98] S. GURUN, C. KRINTZ, AND R. WOLSKI, *Nwslite: a light-weight prediction utility for mobile devices*, in Proceedings of the 2nd international conference on Mobile systems, applications, and services, ACM, 2004, pp. 2–11.

[99] K. HA, G. LEWIS, S. SIMANTA, AND M. SATYANARAYANAN, *Cloud offload in hostile environments*, tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2011.

[100] S. HAN, S. ZHANG, J. CAO, Y. WEN, AND Y. ZHANG, *A resource aware software partitioning algorithm based on mobility constraints in pervasive grid environments*, Future Generation Comp. Syst., 24 (2008), pp. 512–529.

[101] M. A. HASSAN, M. XIAO, Q. WEI, AND S. CHEN, *Help your mobile applications with fog computing*, in 12th Annual IEEE International Conference on Sensing, Communication, and Networking Workshops, SECON Workshops 2015, Seattle, WA, USA, June 22-25, 2015, 2015, pp. 49–54.

[102] ——, *Help your mobile applications with fog computing*, in 12th Annual IEEE International Conference on Sensing, Communication, and Networking Workshops, SECON Workshops 2015, Seattle, WA, USA, June 22-25, 2015, 2015, pp. 49–54.

[103] W. HE, K. YUAN, H. XIAO, AND Z. XU, *A high speed robot vision system with gige vision extension*, in 2011 IEEE International Conference on Mechatronics and Automation, IEEE, 2011, pp. 452–457.

[104] Y. HE, S. ELNIKETY, J. R. LARUS, AND C. YAN, *Zeta: scheduling interactive services with partial execution*, in ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012, 2012, p. 12.

[105] F. HERMENIER, X. LORCA, J. MENAUD, G. MULLER, AND J. L. LAWALL, *Entropy: a consolidation manager for clusters*, in Proceedings of the 5th International Conference on Virtual Execution Environments, VEE 2009, Washington, DC, USA, March 11-13, 2009, 2009, pp. 41–50.

[106] K. HINCKLEY, M. DIXON, R. SARIN, F. GUIMBRETIÈRE, AND R. BALAKRISHNAN, *Codex: a dual screen tablet computer*, in Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009, 2009, pp. 1933–1942.

[107] M. HOGAN, F. LIU, A. SOKOL, AND J. TONG, *Nist cloud computing standards roadmap–version 1.0, natl*, Inst. Stand. Technol. Spec. Publ, (2011), pp. 500–291.

[108] H.-J. HONG, D.-Y. CHEN, C.-Y. HUANG, AND K.-T. CHEN, *Placing Virtual Machines to Optimize Cloud Gaming Experience*, IEEE Transactions on Cloud Computing, 3 (2015), pp. 42–53.

[109] Y.-J. HONG, K. KUMAR, AND Y.-H. LU, *Energy efficient content-based image retrieval for mobile systems*, in Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on, IEEE, 2009, pp. 1673–1676.

[110] G. HU, W. TAY, AND Y. WEN, *Cloud robotics: architecture, challenges and applications*, IEEE Network, 26 (2012), pp. 21–28.

[111] C. HUANG, K. CHEN, D. CHEN, H. HSU, AND C. HSU, *Gaminganywhere: The first open source cloud gaming system*, ACM Trans. on Multimedia Comp., Comm., and App. (TOMM), 10 (2014), p. 10.

[112] D. HUANG, L. YANG, AND S. ZHANG, *Dust: Real-time code offloading system for wearable computing*, in 2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015, 2015, pp. 1–7.

[113] K. HUANG AND V. K. N. LAU, *Enabling wireless power transfer in cellular networks: Architecture, modeling and deployment*, IEEE Trans. Wireless Communications, 13 (2014), pp. 902–912.

[114] G. HUERTA-CANEPA AND D. LEE, *A virtual cloud computing provider for mobile devices*, in Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, ACM, 2010, p. 6.

[115] G. HUMPHREYS, M. HOUSTON, R. NG, R. FRANK, AND S. AHERN, *Chromium: a stream-processing framework for interactive rendering on clusters*, ACM Trans. Graph., 21 (2002), pp. 693–702.

[116] S. HUNG, C. SHIH, J. SHIEH, C. LEE, AND Y. HUANG, *Executing mobile applications on the cloud: Framework and issues*, Computers & Mathematics with Applications, 63 (2012), pp. 573–587.

[117] M. JAIN AND C. DOVROLIS, *End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput*, IEEE/ACM Trans. Netw., 11 (2003), pp. 537–549.

[118] V. JAMWAL AND S. IYER, *Automated refactoring of objects for application partitioning*, in 12th Asia-Pacific Software Engineering Conference (APSEC 2005), 15-17 December 2005, Taipei, Taiwan, 2005, pp. 671–678.

[119] Y. JARARWEH, F. ABABNEH, A. KHREISHAH, F. DOSARI, ET AL., *Scalable cloudlet-based mobile computing model*, Procedia Computer Science, 34 (2014), pp. 434–441.

[120] Y. JARARWEH, L. A. TAWALBEH, F. ABABNEH, AND F. DOSARI, *Resource efficient mobile computing using cloudlet infrastructure*, in IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks, Dalian, MSN 2013, China, December 11-13, 2013, 2013, pp. 373–377.

[121] M. JARSCHEL AND D. SCHLOSSER, *An evaluation of qoe in cloud gaming based on subjective tests*, in Proceedings of the 5th Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011.

[122] D. B. JOHNSON AND D. A. MALTZ, *Dynamic source routing in ad hoc wireless networks*, Mobile computing, (1996), pp. 153–181.

[123] W. JUNIOR, A. FRANÇA, K. DIAS, AND J. N. DE SOUZA, *Supporting mobility-aware computational offloading in mobile cloud environment*, J. Network and Computer Applications, 94 (2017), pp. 93–108.

[124] S. KALASAPUR AND M. KUMAR, *Resource adaptive hierarchical organization in pervasive environments*, in Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International, IEEE, 2009, pp. 1–8.

[125] T. KALLONEN AND J. PORRAS, *Use of distributed resources in mobile environment*, in Software in Telecommunications and Computer Networks, 2006. SoftCOM 2006. International Conference on, IEEE, 2006, pp. 281–285.

[126] T. KÄMÄRÄINEN, M. SIEKKINEN, Y. XIAO, AND A. YLÄ-JÄÄSKI, *Towards pervasive and mobile gaming with distributed cloud infrastructure*, in Proceedings of the 13th ACM Netgames Workshop, 2014.

[127] R. KEMP, N. PALMER, T. KIELMANN, AND H. BAL, *Opportunistic communication for multiplayer mobile gaming: Lessons learned from photoshoot*, in Proceedings of the Second International Workshop on Mobile Opportunistic Networking, ACM, 2010, pp. 182–184.

[128] R. KEMP, N. PALMER, T. KIELMANN, AND H. E. BAL, *Cuckoo: A computation offloading framework for smartphones*, in Mobile Computing, Applications, and Services - Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers, 2010, pp. 59–79.

[129] ——, *The smartphone and the cloud: Power to the user*, in Mobile Computing, Applications, and Services - Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers, 2010, pp. 342–348.

[130] R. KEMP, N. PALMER, T. KIELMANN, F. J. SEINSTRA, N. DROST, J. MAASSEN, AND H. E. BAL, *eyedentify: Multimedia cyber foraging from a smartphone*, in 11th IEEE International Symposium on Multimedia, ISM 2009, San Diego, California, USA, December 14-16, 2009, 2009, pp. 392–399.

[131] M. KHALILBEIGI, R. LISSERMANN, W. KLEINE, AND J. STEIMLE, *Foldme: interacting with double-sided foldable displays*, in Proceedings of the 6th International Conference on Tangible and Embedded Interaction 2012, Kingston, Ontario, Canada, February 19-22, 2012, 2012, pp. 33–40.

[132] A. N. KHAN, M. L. M. KIAH, S. U. KHAN, AND S. A. MADANI, *Towards secure mobile cloud computing: A survey*, Future Generation Comp. Syst., 29 (2013), pp. 1278–1299.

[133] A. R. KHAN, M. OTHMAN, AND S. A. MADANI, *A survey of mobile cloud computing application models*, IEEE Communications Surveys and Tutorials, 16 (2014), pp. 393–413.

[134] M. E. KHODA, M. A. RAZZAQUE, A. ALMOGREN, M. M. HASSAN, A. ALAMRI, AND A. ALELAIWI, *Efficient computation offloading decision in mobile cloud computing over 5g network*, MONET, 21 (2016), pp. 777–792.

[135] D. KIM, M. OK, AND M. PARK, *An intermediate target for quick-relay of remote storage to mobile devices*, in Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part II, 2005, pp. 1035–1044.

[136] J. KIM, B. MOON, AND M. PARK, *Misc: A new availability remote storage system for mobile appliance*, in Networking - ICN 2005, 4th International Conference on Networking, ReunionIsland, France, April 17-21, 2005, Proceedings, Part II, 2005, pp. 504–520.

[137] S. KOSTA, A. AUCINAS, P. HUI, R. MORTIER, AND X. ZHANG, *Unleashing the power of mobile cloud computing using thinkair*, CoRR, abs/1105.3232 (2011).

[138] ——, *Unleashing the power of mobile cloud computing using thinkair*, CoRR, abs/1105.3232 (2011).

[139] ——, *Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading*, in Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012, 2012, pp. 945–953.

[140] S. KOSTA, V. C. PERTA, J. STEFA, P. HUI, AND A. MEI, *Clone2clone (C2C): peer-to-peer networking of smartphones on the cloud*, in 5th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'13, San Jose, CA, USA, June 25-26, 2013, 2013.

[141] B. J. KOT, B. WUENSCHE, J. C. GRUNDY, AND J. G. HOSKING, *Information visualisation utilising 3d computer game engines case study: a source code comprehension tool*, in Proceedings of the 6th ACM Conf. on Computer-Human Interaction (CHI), 2005.

[142] D. KOVACHEV AND R. KLAMMA, *Framework for computation offloading in mobile cloud computing.*, IJIMAI, 1 (2012), pp. 6–15.

[143] D. KOVACHEV, T. YU, AND R. KLAMMA, *Adaptive computation offloading from mobile devices into the cloud*, in 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012, Leganes, Madrid, Spain, July 10-13, 2012, 2012, pp. 784–791.

[144] ——, *Adaptive computation offloading from mobile devices into the cloud*, in 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012, Leganes, Madrid, Spain, July 10-13, 2012, 2012, pp. 784–791.

[145] U. KREMER, J. HICKS, AND J. M. REHG, *A compilation framework for power and energy management on mobile computers*, in Languages and Compilers for Parallel Computing, 14th International Workshop, LCPC 2001, Cumberland Falls, KY, USA, August 1-3, 2001. Revised Papers, 2001, pp. 115–131.

[146] M. D. KRISTENSEN, *Enabling cyber foraging for mobile devices*, in Proceedings of the 5th MiNEMA Workshop: Middleware for Network Eccentric and Mobile Applications, 2007, pp. 32–36.

[147] M. D. KRISTENSEN, *Scavenger: Transparent development of efficient cyber foraging applications*, in Eigth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2010, March 29 - April 2, 2010, Mannheim, Germany, 2010, pp. 217–226.

[148] J. KROGSTIE, *Requirement engineering for mobile information systems*, in Proc. of International Workshop on Requirements Engineering: Foundation for Software Quality (Interlaken, Switzerland), 2001.

[149] A. KSENTINI, T. TALEB, AND F. MESSAOUDI, *A lisp-based implementation of follow me cloud*, IEEE Access, 2 (2014), pp. 1340–1347.

[150] K. KUMAR, J. LIU, Y. LU, AND B. K. BHARGAVA, *A survey of computation offloading for mobile systems*, MONET, 18 (2013), pp. 129–140.

[151] K. KUMAR AND Y. LU, *Cloud computing for mobile users: Can offloading computation save energy?*, IEEE Computer, 43 (2010), pp. 51–56.

[152] K. LAI AND M. BAKER, *Measuring link bandwidths using a deterministic model of packet delay*, in SIGCOMM, 2000, pp. 283–294.

[153] D. Q. M. LANTIN AND A. G. S. ARDEN, *High frame rate (hfr), a white paper*.

[154] B.-D. LEE, *A framework for seamless execution of mobile applications in the cloud*, in Recent advances in computer science and information engineering, Springer, 2012, pp. 145–153.

[155] G. LEE, H. PARK, S. HEO, K. CHANG, H. LEE, AND H. KIM, *Architecture-aware automatic computation offload for native applications*, in Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015, 2015, pp. 521–532.

[156] J.-S. LEE, Y.-W. SU, AND C.-C. SHEN, *A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi*, in Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE, Ieee, 2007, pp. 46–51.

[157] K. LEE, D. CHU, AND E. CUERVO, *Demo: Delorean: using speculation to enable low-latency continuous interaction for mobile cloud gaming*, in Proceedings of the 12th ACM MobiSys Conference, 2014.

[158] K. LEE AND I. SHIN, *User mobility-aware decision making for mobile computation offloading*, in 1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA 2013, Taipei, Taiwan, August 19-20, 2013, 2013, pp. 116–119.

[159] S. LEE, I. PEFKIANAKIS, A. MEYERSON, S. XU, AND S. LU, *Proportional fair frequency-domain packet scheduling for 3gpp LTE uplink*, in INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil, 2009, pp. 2611–2615.

[160] Y. LEE, K. CHEN, H. SU, AND C. LEI, *Are all games equally cloud-gaming-friendly? an electromyographic approach*, in Proceedings of the 11th ACM Netgames Workshop, 2012.

[161] L. LEI, Z. ZHONG, K. ZHENG, J. CHEN, AND H. MENG, *Challenges on wireless heterogeneous networks for mobile cloud computing*, IEEE Wireless Commun., 20 (2013), pp. 1–0.

[162] V. LENDERS, J. WAGNER, AND M. MAY, *Analyzing the impact of mobility in ad hoc networks*, in Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality, ACM, 2006, pp. 39–46.

[163] M. LEWIS AND J. JACOBSON, *Game engines*, Communications of the ACM, 45 (2002), p. 27.

[164] C. LI AND L. LI, *Energy constrained resource allocation optimization for mobile grids*, J. Parallel Distrib. Comput., 70 (2010), pp. 245–258.

[165] Y. LI, X. TANG, AND W. CAI, *Play request dispatching for efficient virtual machine usage in cloud gaming*, IEEE Trans. Circuits Syst. Video Techn., 25 (2015), pp. 2052–2063.

[166] Z. LI, C. WANG, AND R. XU, *Computation offloading to save energy on handheld devices: a partition scheme*, in Proceedings of the 2001 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES 2001, Atlanta, Georgia, USA, November 16-17, 2001, 2001, pp. 238–246.

[167] ——, *Task allocation for distributed multimedia processing on wirelessly networked handheld devices*, in 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), 15-19 April 2002, Fort Lauderdale, FL, USA, CD-ROM/Abstracts Proceedings, 2002.

[168] Z. LI AND R. XU, *Energy impact of secure computation on a handheld device*, in Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on, IEEE, 2002, pp. 109–117.

[169] J. LIU, Y. MAO, J. ZHANG, AND K. B. LETAIEF, *Delay-optimal computation task scheduling for mobile-edge computing systems*, CoRR, abs/1604.07525 (2016).

[170] Q. LIU, X. JIAN, J. HU, H. ZHAO, AND S. ZHANG, *An optimized solution for mobile environment using mobile cloud computing*, in Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on, IEEE, 2009, pp. 1–5.

[171] Y. LIU, S. DEY, AND Y. LU, *Enhancing video encoding for cloud gaming using rendering information*, IEEE Trans. Circuits Syst. Video Techn., 25 (2015), pp. 1960–1974.

[172] Y. LIU, Z. QIN, AND C. ZHAO, *Autocharge: Automatically charge smartphones using a light beam*, (2015).

[173] H. LUO, S. CI, D. WU, J. WU, AND H. TANG, *Quality-driven cross-layer optimized video delivery over LTE*, IEEE Communications Magazine, 48 (2010), pp. 102–109.

[174] M. LUO AND M. CLAYPOOL, *Uniquitous: Implementation and evaluation of a cloud-based game system in unity*, in Proceedings of IEEE GEM Conf., 2015.

[175] X. LUO, *From augmented reality to augmented computing: A look at cloud-mobile convergence*, in Ubiquitous Virtual Reality, 2009. ISUVR'09. International Symposium on, IEEE, 2009, pp. 29–32.

[176] R. K. K. MA, K. T. LAM, AND C. WANG, *excloud: Transparent runtime support for scaling mobile applications in cloud*, in 2011 International Conference on Cloud and Service Computing, CSC 2011, Hong Kong, December 12-14, 2011, 2011, pp. 103–110.

[177] C. M. S. MAGURAWALAGE, K. YANG, AND K. WANG, *Aqua computing: Coupling computing and communications*, CoRR, abs/1510.07250 (2015).

[178] S. E. MAHMOODI AND K. P. S. SUBBALAKSHMI, *A time-adaptive heuristic for cognitive cloud offloading in multi-rat enabled wireless devices*, IEEE Trans. Cogn. Comm. & Networking, 2 (2016), pp. 194–207.

[179] S. E. MAHMOODI, R. UMA, AND K. SUBBALAKSHMI, *Optimal joint scheduling and cloud offloading for mobile applications*, IEEE Transactions on Cloud Computing, (2016).

[180] P. MAKRIS, D. N. SKOUTAS, AND C. SKIANIS, *A survey on context-aware mobile and wireless networking: On networking and computing environments' integration*, IEEE Communications Surveys and Tutorials, 15 (2013), pp. 362–386.

[181] A. MANJUNATHA, A. RANABAHU, A. P. SHETH, AND K. THIRUNARAYAN, *Power of clouds in your pocket: An efficient approach for cloud mobile hybrid application development*, in Cloud Computing, Second International Conference, CloudCom 2010, November 30 - December 3, 2010, Indianapolis, Indiana, USA, Proceedings, 2010, pp. 496–503.

[182] S. S. MANVI AND G. K. SHYAM, *Resource management for infrastructure as a service (iaas) in cloud computing: A survey*, J. Network and Computer Applications, 41 (2014), pp. 424–440.

[183] Y. MAO, J. ZHANG, S. SONG, AND K. B. LETAIEF, *Power-delay tradeoff in multi-user mobile-edge computing systems*, in 2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016, 2016, pp. 1–6.

[184] V. MARCH, Y. GU, E. LEONARDI, G. GOH, M. KIRCHBERG, AND B. LEE, *µcloud: Towards a new paradigm of rich mobile applications*, in Proceedings of the 2nd International Conference on Ambient Systems, Networks and Technologies (ANT 2011), the 8th International Conference on Mobile Web Information Systems (MobiWIS-2011), Niagara Falls, Ontario, Canada, September 19-21, 2011, 2011, pp. 618–624.

[185] S. MARKS, J. A. WINDSOR, AND B. WÜNSCHE, *Evaluation of game engines for simulated surgical training*, in Proc of the 5th ACM Int. Conf. on Computer Graphics and Interactive Techniques (Graphite), 2007.

[186] B. MELANDER, M. BJÖRKMAN, AND P. GUNNINGBERG, *A new end-to-end probing and analysis method for estimating bandwidth bottlenecks*, in Proceedings of the Global Telecommunications Conference, 2000. GLOBECOM 2000, San Francisco, CA, USA, 27 November - 1 December 2000, 2000, pp. 415–420.

[187] F. MESSAOUDI, A. KSENTINI, AND P. BERTIN, *Toward a mobile gaming based-computation offloading*, in Proceedings of IEEE International Conference on Communications, ICC 2018, Kansas City, MO, USA, 20-24 Mai 2018, 2007, pp. 1821–1826.

[188] ——, *On using edge computing for computation offloading in mobile network*, in 2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, December 4-8, 2017, 2017, pp. 1–7.

[189] F. MESSAOUDI, A. KSENTINI, G. SIMON, AND P. BERTIN, *Performance analysis of game engines on mobile and fixed devices*, ACM Trans. Multimedia Comput. Commun. Appl., 13 (2017), pp. 57:1–57:28.

[190] ——, *Performance analysis of game engines on mobile and fixed devices*, ACM Trans. Multimedia Comput. Commun. Appl., 13 (2017), pp. 57:1–57:28.

[191] F. MESSAOUDI, G. SIMON, AND A. KSENTINI, *Dissecting games engines: The case of unity3d*, in 2015 International Workshop on Network and Systems Support for Games, NetGames 2015, Zagreb, Croatia, December 3-4, 2015, 2015, pp. 1–6.

[192] F. MESSAOUDI, G. SIMON, AND A. KSENTINI, *Dissecting games engines: The case of unity3d*, in Network and Systems Support for Games (NetGames), 2015 International Workshop on, Dec 2015, pp. 1–6.

[193] A. MESSER, I. GREENBERG, P. BERNADAT, D. S. MILOJICIC, D. CHEN, T. J. GIULI, AND X. GU, *Towards a distributed platform for resource-constrained devices*, in ICDCS, 2002, pp. 43–51.

[194] I. MILCHTAICH, *Congestion games with player-specific payoff functions*, Games and economic behavior, 13 (1996), pp. 111–124.

[195] B. A. MILLER, T. NIXON, C. TAI, AND M. D. WOOD, *Home networking with universal plug and play*, IEEE Communications Magazine, 39 (2001), pp. 104–109.

[196] K. MITRA, S. SAGUNA, C. ÅHLUND, AND D. GRANLUND, *M^2c^2: A mobility management system for mobile cloud computing*, in 2015 IEEE Wireless Communications and Networking Conference, WCNC 2015, New Orleans, LA, USA, March 9-12, 2015, 2015, pp. 1608–1613.

[197] M. MOWBRAY AND S. PEARSON, *A client-based privacy manager for cloud computing*, in Proceedings of the 4th International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2009), June 15-19, 2009, Dublin, Ireland, 2009, p. 5.

[198] A. F. MURARASU AND T. MAGEDANZ, *Mobile middleware solution for automatic reconfiguration of applications*, in Sixth International Conference on Information Technology: New Generations, ITNG 2009, Las Vegas, Nevada, 27-29 April 2009, 2009, pp. 1049–1055.

[199] V. NAE, A. IOSUP, AND R. PRODAN, *Dynamic resource provisioning in massively multiplayer online games*, IEEE Trans. Parallel Distrib. Syst., 22 (2011), pp. 380–395.

[200] D. NARAYANAN, J. FLINN, AND M. SATYANARAYANAN, *Using history to improve mobile application adaptation*, in 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000), 7-8 December 2000, Monterey, CA, USA, 2000, p. 31.

[201] R. M. NASIRI, J. WANG, A. REHMAN, AND S. WANG, *Perceptual quality assessment of high frame rate video*, in Proceedings of the 17th IEEE International Workshop on Multimedia Signal Processing (MMSP), 2015.

[202] M. E. NASR, M. AL-SAATI, AND S. NIEDENTHAL, *Assassin's creed: A multi-cultural read*, (2008).

[203] M. NASSERI, M. ALAM, AND R. C. G. II, *MDP based optimal policy for collaborative processing using mobile cloud computing*, in IEEE 2nd International Conference on Cloud Networking, CloudNet 2013, San Francisco, CA, USA, November 11-13, 2013, 2013, pp. 123–129.

[204] F. NAZIR, J. MA, AND A. SENEVIRATNE, *Time critical content delivery using predictable patterns in mobile social networks*, in Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada, August 29-31, 2009, 2009, pp. 1066–1073.

[205] R. NEWTON, S. TOLEDO, L. GIROD, H. BALAKRISHNAN, AND S. MADDEN, *Wishbone: Profile-based partitioning for sensornet applications*, in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA, 2009, pp. 395–408.

[206] B. NG, A. SI, R. W. H. LAU, AND F. W. B. LI, *A multi-server architecture for distributed virtual walkthrough*, in Proc. of the ACM Symp. on Virtual Reality Software and Tech. VRST, 2002, pp. 163–170.

[207] R. NIEMANN AND P. MARWEDEL, *An algorithm for hardware/software partitioning using mixed integer linear programming*, Design Autom. for Emb. Sys., 2 (1997), pp. 165–193.

[208] Y. NIMMAGADDA, K. KUMAR, Y.-H. LU, AND C. G. LEE, *Real-time moving object recognition and tracking using computation offloading*, in Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE, 2010, pp. 2449–2455.

[209] J. NIU, W. SONG, AND M. ATIQUZZAMAN, *Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications*, J. Network and Computer Applications, 37 (2014), pp. 334–347.

[210] J. OBERHEIDE, K. VEERARAGHAVAN, E. COOKE, J. FLINN, AND F. JAHANIAN, *Virtualized in-cloud security services for mobile devices*, in Proceedings of the First Workshop on Virtualization in Mobile Computing, Breckenridge, CO, USA, June 17, 2008, 2008, pp. 31–35.

[211] R. O'DONNELL, *Prolog to energy harvesting from human and machine motion for wireless electronic devices*, Proceedings of the IEEE, 96 (2008), pp. 1455–1456.

[212] J. OH, S. LEE, AND E. LEE, *An adaptive mobile system using mobile grid computing in wireless network*, in Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part V, 2006, pp. 49–57.

[213] K. J. O'HARA, R. NATHUJI, H. RAJ, K. SCHWAN, AND T. R. BALCH, *Autopower: Toward energy-aware software systems for distributed mobile robots*, in Proceedings of the 2006

IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA, 2006, pp. 2757–2762.

[214] M. OK, D. KIM, AND M. PARK, *Ubiqstor: A remote storage service for mobile devices*, in Parallel and Distributed Computing: Applications and Technologies, 5th International Conference, PDCAT 2004, Singapore, December 8-10, 2004, Proceedings, 2004, pp. 685–688.

[215] ——, *Ubiqstor: Server and proxy for remote storage of mobile devices*, in Emerging Directions in Embedded and Ubiquitous Computing, EUC 2006 Workshops: NCUS, SecUbiq, USN, TRUST, ESO, and MSA, Seoul, Korea, August 1-4, 2006, Proceedings, 2006, pp. 22–31.

[216] M. OTHMAN AND S. HAILES, *Power conservation strategy for mobile computers using load sharing*, Mobile Computing and Communications Review, 2 (1998), pp. 44–51.

[217] S. OU, K. YANG, AND A. LIOTTA, *An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems*, in 4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006), 13-17 March 2006, Pisa, Italy, 2006, pp. 116–125.

[218] S. OU, K. YANG, A. LIOTTA, AND L. HU, *Performance analysis of offloading systems in mobile wireless environments*, in Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007, 2007, pp. 1821–1826.

[219] S. OU, K. YANG, AND Q. ZHANG, *An efficient runtime offloading approach for pervasive services*, in Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE, vol. 4, IEEE, 2006, pp. 2229–2234.

[220] S. OU, K. YANG, AND Q. ZHANG, *An efficient runtime offloading approach for pervasive services*, in IEEE Wireless Communications and Networking Conference, WCNC 2006, 3-6 April 2006, Las Vegas, Nevada, USA, 2006, pp. 2229–2234.

[221] E. PARK, H. SHIN, AND S. J. KIM, *Selective grid access for energy-aware mobile computing*, in International Conference on Ubiquitous Intelligence and Computing, Springer, 2007, pp. 798–807.

[222] S. PARK, B.-S. MOON, AND M.-S. PARK, *Design and implementation of iscsi-based remote storage system for mobile appliance*, Proc. IEEE HEALTHCOM'05, (2003), pp. 236–240.

[223] L. D. PEDROSA, N. KOTHARI, R. GOVINDAN, J. VAUGHAN, AND T. MILLSTEIN, *The case for complexity prediction in automatic partitioning of cloud-enabled mobile applications*, Small, 20 (2012), p. 25.

[224] K. PETTIS AND R. C. HANSEN, *Profile guided code positioning*, in Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation (PLDI), White Plains, New York, USA, June 20-22, 1990, 1990, pp. 16–27.

[225] H. PFISTER, M. ZWICKER, J. VAN BAAR, AND M. GROSS, *Surfels: Surface elements as rendering primitives*, in Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335–342.

[226] M. PHILIPPSEN AND M. ZENGER, *Javaparty - transparent remote objects in java*, Concurrency Practice and Experience, 9 (1997), pp. 1225–1242.

[227] G. PORTOKALIDIS, *Using virtualisation to protect against zero-day attacks*, (2010).

[228] R. PRASAD, C. DOVROLIS, M. MURRAY, AND K. CLAFFY, *Bandwidth estimation: metrics, measurement techniques, and tools*, IEEE Network, 17 (2003), pp. 27–35.

[229] P. QUAX, A. BEZNOSYK, W. VANMONTFORT, AND R. MARX, *An evaluation of the impact of game genre on user experience in cloud gaming*, in Proc. of the IEEE Int. Games Innov. Conf. (IGIC), 2013, pp. 216–221.

[230] M. RA, B. PRIYANTHA, A. KANSAL, AND J. LIU, *Improving energy efficiency of personal sensing applications with heterogeneous multi-processors*, in The 2012 ACM Conference on Ubiquitous Computing, Ubicomp '12, Pittsburgh, PA, USA, September 5-8, 2012, 2012, pp. 1–10.

[231] M. RA, A. SHETH, L. B. MUMMERT, P. PILLAI, D. WETHERALL, AND R. GOVINDAN, *Odessa: enabling interactive perception applications on mobile devices*, in Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Bethesda, MD, USA, June 28 - July 01, 2011, 2011, pp. 43–56.

[232] P. RAAD, G. COLOMBO, D. P. CHI, S. SECCI, A. CIANFRANI, P. GALLARD, AND G. PUJOLLE, *Achieving sub-second downtimes in internet-wide virtual machine live migrations in LISP networks*, in 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 27-31, 2013, 2013, pp. 286–293.

[233] K. RAAEN, R. EG, AND C. GRIWODZ, *Can gamers detect cloud delay?*, in Proceedings of the 13th ACM/IEEE Netgames Workshop, 2014.

[234] K. RAAEN AND A. PETLUND, *How Much Delay is There Really in Current Games?*, in Proceedings of the 6th ACM Multimedia Systems (MMSys) Conference, 2015.

[235] M. R. RAHIMI, N. VENKATASUBRAMANIAN, S. MEHROTRA, AND A. V. VASILAKOS, *Mapcloud: Mobile applications on an elastic and scalable 2-tier cloud architecture*, in IEEE Fifth International Conference on Utility and Cloud Computing, UCC 2012, Chicago, IL, USA, November 5-8, 2012, 2012, pp. 83–90.

[236] A. RAI, R. BHAGWAN, AND S. GUHA, *Generalized resource allocation for the cloud*, in ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012, 2012, p. 15.

[237] H. RIM, S. KIM, Y. KIM, AND H. HAN, *Transparent method offloading for slim execution*, in Wireless Pervasive Computing, 2006 1st International Symposium on, IEEE, 2006, pp. 1–6.

[238] B. P. RIMAL, E. CHOI, AND I. LUMB, *A taxonomy and survey of cloud computing systems*, in International Conference on Networked Computing and Advanced Information Management, NCM 2009, Fifth International Joint Conference on INC, IMS and IDC: INC 2009: International Conference on Networked Computing, IMS 2009: International Conference on Advanced Information Management and Service, IDC 2009: International Conference on Digital Content, Multimedia Technology and its Applications, Seoul, Korea, August 25-27, 2009, 2009, pp. 44–51.

[239] P. RONG AND M. PEDRAM, *Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach*, in Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003, 2003, pp. 906–911.

[240] ——, *A markovian decision-based approach for extending the lifetime of a network of battery-powered mobile devices by remote processing*, J. Low Power Electronics, 6 (2010), pp. 227–239.

[241] A. RUDENKO, P. REIHER, G. J. POPEK, AND G. H. KUENNING, *The remote processing framework for portable computer power saving*, in Proceedings of the 1999 ACM symposium on Applied computing, ACM, 1999, pp. 365–372.

[242] A. RUDENKO, P. L. REIHER, G. J. POPEK, AND G. H. KUENNING, *Saving portable computer battery power through remote process execution*, Mobile Computing and Communications Review, 2 (1998), pp. 19–26.

[243] A. SACKL, R. SCHATZ, AND T. HOSSFELD, *QoE Management made uneasy: The case of Cloud Gaming*, in Proceedings of the IEEE International Conference on Communications Workshops (ICC), 2016.

[244] T. SAITO AND T. TAKAHASHI, *Comprehensible rendering of 3-d shapes*, in Proceedings of the 17th ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), 1990, pp. 197–206.

[245] Z. SANAEI, S. ABOLFAZLI, A. GANI, AND R. BUYYA, *Heterogeneity in mobile cloud computing: Taxonomy and open challenges*, IEEE Communications Surveys and Tutorials, 16 (2014), pp. 369–392.

[246] Z. SANAEI, S. ABOLFAZLI, A. GANI, AND M. SHIRAZ, *Sami: Service-based arbitrated multi-tier infrastructure for mobile cloud computing*, in Communications in China Workshops (ICCC), 2012 1st IEEE International Conference on, IEEE, 2012, pp. 14–19.

[247] S. SARDELLITTI, G. SCUTARI, AND S. BARBAROSSA, *Joint optimization of radio and computational resources for multicell mobile-edge computing*, IEEE Trans. Signal and Information Processing over Networks, 1 (2015), pp. 89–103.

[248] M. SATYANARAYANAN, *Pervasive computing: vision and challenges*, IEEE Personal Commun., 8 (2001), pp. 10–17.

[249] M. SATYANARAYANAN, P. BAHL, R. CACERES, AND N. DAVIES, *The case for vm-based cloudlets in mobile computing*, IEEE pervasive Computing, 8 (2009).

[250] M. SATYANARAYANAN, P. BAHL, R. CÁCERES, AND N. DAVIES, *The case for vm-based cloudlets in mobile computing*, IEEE Pervasive Computing, 8 (2009), pp. 14–23.

[251] M. SATYANARAYANAN, Z. CHEN, K. HA, W. HU, W. RICHTER, AND P. PILLAI, *Cloudlets: at the leading edge of mobile-cloud convergence*, in Proceedings of MobiCASE, 2014.

[252] S. SCHNEEGANS, F. LAUER, A. BERNSTEIN, AND A. SCHOLLMEYER, *guacamole - an extensible scene graph and rendering framework based on deferred shading*, in Proceedings of the 7th IEEE Workshop on Software Engineering and Architectures for Realtime Interactive Systems, SEARIS, 2014.

[253] S. SCHWARZ, C. MEHLFÜHRER, AND M. RUPP, *Low complexity approximate maximum throughput scheduling for lte*, in Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on, IEEE, 2010, pp. 1563–1569.

[254] M. SEMSARZADEH, A. YASSINE, AND S. SHIRMOHAMMADI, *Video encoding acceleration in cloud gaming*, IEEE Trans. Circuits Syst. Video Techn., 25 (2015), pp. 1975–1987.

[255] B. SESHASAYEE, R. NATHUJI, AND K. SCHWAN, *Energy-aware mobile service overlays: Cooperative dynamic power management in distributed mobile systems*, in Autonomic Computing, 2007. ICAC'07. Fourth International Conference on, IEEE, 2007, pp. 6–12.

[256] R. SHEA, D. FU, AND J. LIU, *Cloud gaming: Understanding the support from advanced virtualization and hardware*, IEEE Trans. on Circuits and Systems for Video Tech., 25 (2015), pp. 2026–2037.

[257] R. SHEA AND J. LIU, *On gpu pass-through performance for cloud gaming: Experiments and analysis*, in Proceedings of 12th IEEE/ACM Netgames Workshop, 2013.

[258] Z. SHEN, S. SUBBIAH, X. GU, AND J. WILKES, *Cloudscale: elastic resource scaling for multi-tenant cloud systems*, in ACM Symposium on Cloud Computing in conjunction with SOSP 2011, SOCC '11, Cascais, Portugal, October 26-28, 2011, 2011, p. 5.

[259] Z. SHEN, S. SUBBIAH, X. GU, AND J. WILKES, *Cloudscale: elastic resource scaling for multi-tenant cloud systems*, in Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 5.

[260] C. SHI, K. HABAK, P. PANDURANGAN, M. H. AMMAR, M. NAIK, AND E. W. ZEGURA, *COSMOS: computation offloading as a service for mobile devices*, in The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014, 2014, pp. 287–296.

[261] C. SHI, P. PANDURANGAN, K. NI, J. YANG, M. AMMAR, M. NAIK, AND E. ZEGURA, *Ic-cloud: Computation offloading to an intermittently-connected cloud*, tech. rep., Georgia Institute of Technology, 2013.

[262] J. SHUJA, A. GANI, K. BILAL, A. UR REHMAN KHAN, S. A. MADANI, S. U. KHAN, AND A. Y. ZOMAYA, *A survey of mobile device virtualization: Taxonomy and state of the art*, ACM Comput. Surv., 49 (2016), pp. 1:1–1:36.

[263] J. SHUJA, A. GANI, A. NAVEED, E. AHMED, AND C. HSU, *Case of ARM emulation optimization for offloading mechanisms in mobile cloud computing*, Future Generation Comp. Syst., 76 (2017), pp. 407–417.

[264] P. SIMOENS, F. D. TURCK, B. DHOEDT, AND P. DEMEESTER, *Remote display solutions for mobile cloud computing*, IEEE Computer, 44 (2011), pp. 46–53.

[265] K. SINHA AND M. KULKARNI, *Techniques for fine-grained, multi-site computation offloading*, in Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2011, pp. 184–194.

[266] K. SINHA AND M. KULKARNI, *Techniques for fine-grained, multi-site computation offloading*, in 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011, 2011, pp. 184–194.

[267] S. SIVAVAKEESAR, O. F. GONZALEZ, AND G. PAVLOU, *Service discovery strategies in ubiquitous communication environments*, IEEE Communications Magazine, 44 (2006), pp. 106–113.

[268] J. SMED, T. KAUKORANTA, AND H. HAKONEN, *Aspects of networking in multiplayer computer games*, The Electronic Library, 20 (2002), pp. 87–97.

[269] M. SMIT, M. SHTERN, B. SIMMONS, AND M. LITOIU, *Partitioning applications for hybrid and federated clouds*, in Center for Advanced Studies on Collaborative Research, CASCON '12, Toronto, ON, Canada, November 5-7, 2012, 2012, pp. 27–41.

[270] M. SOOKHAK, H. TALEBIAN, E. AHMED, A. GANI, AND M. K. KHAN, *A review on remote data auditing in single cloud server: Taxonomy and open issues*, J. Network and Computer Applications, 43 (2014), pp. 121–141.

[271] T. SOYATA, R. MURALEEDHARAN, S. AMES, J. LANGDON, C. FUNAI, M. KWON, AND W. HEINZELMAN, *Combat: mobile cloud-based compute/communications infrastructure for battlefield applications*, in Proceedings of SPIE, vol. 8403, 2012, pp. 84030K–84030K.

[272] T. SOYATA, R. MURALEEDHARAN, C. FUNAI, M. KWON, AND W. B. HEINZELMAN, *Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture*, in 2012 IEEE Symposium on Computers and Communications, ISCC 2012, Cappadocia, Turkey, July 1-4, 2012, 2012, pp. 59–66.

[273] M. STOER AND F. WAGNER, *A simple min-cut algorithm*, J. ACM, 44 (1997), pp. 585–591.

[274] O. STORZ, A. FRIDAY, AND N. DAVIES, *Towards 'ubiquitous' ubiquitous computing: an alliance with 'the grid'*, in System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003), 2003.

[275] Y. SU AND J. FLINN, *Slingshot: deploying stateful services in wireless hotspots*, in Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys 2005, Seattle, Washington, USA, June 6-8, 2005, 2005, pp. 79–92.

[276] M. SUZNJEVIC, J. SALDANA, AND M. MATIJASEVIC, *Analyzing the effect of TCP and Server Population on Massively Multiplayer Games*, International Journal of Computer Games Technology, 2014 (2014), p. 2.

[277] E. TILEVICH AND Y. SMARAGDAKIS, *J-orchestra: Automatic java application partitioning*, in ECOOP 2002 - Object-Oriented Programming, 16th European Conference, Malaga, Spain, June 10-14, 2002, Proceedings, 2002, pp. 178–204.

[278] ——, *J-orchestra: Enhancing java programs with distribution capabilities*, ACM Trans. Softw. Eng. Methodol., 19 (2009), pp. 1:1–1:40.

[279] N. TOLIA, D. G. ANDERSEN, AND M. SATYANARAYANAN, *Quantifying interactive user experience on thin clients*, Computer, 39 (2006), pp. 46–52.

[280] M.-K. TSAI, Y.-C. LEE, C.-H. LU, M.-H. CHEN, T.-Y. CHOU, AND N.-J. YAU, *Integrating geographical information and augmented reality techniques for mobile escape guidelines on nuclear accident sites*, Journal of environmental radioactivity, 109 (2012), pp. 36–44.

[281] J. TULIP, J. BEKKEMA, AND K. NESBITT, *Multi-threaded game engine design*, in Proceedings of the Interactive Entertainment conf (IE), 2006.

[282] J. VEIZADES, E. GUTTMAN, C. E. PERKINS, AND S. KAPLAN, *Service location protocol*, RFC, 2165 (1997), pp. 1–72.

[283] T. VERBELEN, P. SIMOENS, F. DE TURCK, AND B. DHOEDT, *Cloudlets: Bringing the cloud to the mobile user*, in Proceedings of the third ACM workshop on Mobile cloud computing and services, ACM, 2012, pp. 29–36.

[284] ——, *Cloudlets: bringing the cloud to the mobile user*, in 3rd ACM Workshop on Mobile Cloud Computing and Services, Proceedings, Ghent University, Department of Information technology, 2012, pp. 29–35.

[285] T. VERBELEN, P. SIMOENS, F. D. TURCK, AND B. DHOEDT, *AIOLOS: middleware for improving mobile application performance through cyber foraging*, Journal of Systems and Software, 85 (2012), pp. 2629–2639.

[286] ——, *Leveraging cloudlets for immersive collaborative applications*, IEEE Pervasive Computing, 12 (2013), pp. 30–38.

[287] T. VERBELEN, T. STEVENS, P. SIMOENS, F. D. TURCK, AND B. DHOEDT, *Dynamic deployment and quality adaptation for mobile augmented reality applications*, Journal of Systems and Software, 84 (2011), pp. 1871–1882.

[288] ——, *Dynamic deployment and quality adaptation for mobile augmented reality applications*, Journal of Systems and Software, 84 (2011), pp. 1871–1882.

[289] M. VÉRON, O. MARIN, AND S. MONNET, *Matchmaking in multi-player on-line games: studying user traces to improve the user experience*, in Proceedings of the 13th ACM/IEEE Netgames Workshop, 2014.

[290] C. WANG AND Z. LI, *A computation offloading scheme on handheld devices*, Journal of Parallel and Distributed Computing, 64 (2004), pp. 740–746.

[291] C. WANG AND Z. LI, *Parametric analysis for adaptive computation offloading*, in Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004, Washington, DC, USA, June 9-11, 2004, 2004, pp. 119–130.

[292] C. WANG, K. REN, W. LOU, AND J. LI, *Toward publicly auditable secure cloud data storage services*, IEEE Network, 24 (2010), pp. 19–24.

[293] C. WANG, F. R. YU, C. LIANG, Q. CHEN, AND L. TANG, *Joint computation offloading and interference management in wireless cellular networks with mobile edge computing*, IEEE Trans. Vehicular Technology, 66 (2017), pp. 7432–7445.

[294] G. WANG, Q. LIU, AND J. WU, *Hierarchical attribute-based encryption for fine-grained access control in cloud storage services*, in Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, 2010, pp. 735–737.

[295] K. WANG, J. RAO, AND C. XU, *Rethink the virtual machine template*, in Proceedings of the 7th International Conference on Virtual Execution Environments, VEE 2011, Newport Beach, CA, USA, March 9-11, 2011 (co-located with ASPLOS 2011), 2011, pp. 39–50.

[296] L. WANG AND M. FRANZ, *Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives*, in 14th International Conference on Parallel and Distributed Systems, ICPADS 2008, Melbourne, Victoria, Australia, December 8-10, 2008, 2008, pp. 369–376.

[297] Q. WANG, C. WANG, J. LI, K. REN, AND W. LOU, *Enabling public verifiability and data dynamics for storage security in cloud computing*, in Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings, 2009, pp. 355–370.

[298] W. WANG, Z. LI, R. OWENS, AND B. K. BHARGAVA, *Secure and efficient access to outsourced data*, in Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009, 2009, pp. 55–66.

[299] H. WATANABE, T. OHIGASHI, T. KONDO, K. NISHIMURA, AND R. AIBARA, *A performance improvement method for the global live migration of virtual machine with ip mobility*, in Proc. 5th Int. Conf. on Mobile Computing and Ubiquitous Networking, Seattle, 2010, pp. 194–199.

[300] S. D. WEBB, S. SOH, AND W. LAU, *Enhanced mirrored servers for network games*, in Proceedings of the 6th ACM Netgames Workshop, 2007.

[301] L. C. WOLF, ed., *Proceedings of the 1st Workshop on Network and System Support for Games, NETGAMES 2002, Braunschweig, Germany, April 16-17, 2002, 2003*, ACM, 2002.

[302] R. WOLSKI, S. GURUN, C. KRINTZ, AND D. NURMI, *Using bandwidth data to make computation offloading decisions*, in 22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008, 2008, pp. 1–8.

[303] H. WU, K. WOLTER, AND A. GRAZIOLI, *Cloudlet-based mobile offloading systems: a performance analysis*, in IFIP WG 7.3 Performance 2013 31 st International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2013 Student Poster Abstracts September 24-26, Vienna, Austria, 2013.

[304] C. XIAN, Y. LU, AND Z. LI, *Adaptive computation offloading for energy conservation on battery-powered systems*, in 13th International Conference on Parallel and Distributed Systems, ICPADS 2007, Hsinchu, Taiwan, December 5-7, 2007, 2007, pp. 1–8.

[305] Z. XIAO AND Y. XIAO, *Security and privacy in cloud computing*, IEEE Communications Surveys and Tutorials, 15 (2013), pp. 843–859.

[306] A. YAHYAVI, K. HUGUENIN, J. GASCON-SAMSON, J. KIENZLE, AND B. KEMME, *Watchmen: Scalable cheat-resistant support for distributed multi-player online games*, in Proc. of the 33rd IEEE Int. Conf. on Dist. Comp. Sys. (ICDCS), 2013.

[307] A. YAHYAVI AND B. KEMME, *Peer-to-peer architectures for massively multiplayer online games: A survey*, ACM Comput. Surv., 46 (2013), p. 9.

[308] J. YANG, H. WANG, J. WANG, C. TAN, AND D. YU, *Provable data possession of resource-constrained mobile devices in cloud computing*, JNW, 6 (2011), pp. 1033–1040.

[309] K. YANG AND X. JIA, *An efficient and secure dynamic auditing protocol for data storage in cloud computing*, IEEE Trans. Parallel Distrib. Syst., 24 (2013), pp. 1717–1726.

[310] K. YANG, S. OU, AND H.-H. CHEN, *On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications*, IEEE communications magazine, 46 (2008).

[311] L. YANG, J. CAO, Y. YUAN, T. LI, A. HAN, AND A. CHAN, *A framework for partitioning and execution of data stream applications in mobile cloud computing*, ACM SIGMETRICS Performance Evaluation Review, 40 (2013), pp. 23–32.

[312] L. YANG, J. CAO, Y. YUAN, T. LI, A. HAN, AND A. T. S. CHAN, *A framework for partitioning and execution of data stream applications in mobile cloud computing*, SIGMETRICS Performance Evaluation Review, 40 (2013), pp. 23–32.

[313] K. YAP, M. KOBAYASHI, R. SHERWOOD, T. HUANG, M. CHAN, N. HANDIGOL, AND N. MCKEOWN, *Openroads: empowering research in mobile networks*, Computer Communication Review, 40 (2010), pp. 125–126.

[314] S. YI, Z. HAO, Z. QIN, AND Q. LI, *Fog computing: Platform and applications*, in Third IEEE Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015, Washington, DC, USA, November 12-13, 2015, 2015, pp. 73–78.

[315] C. YOU AND K. HUANG, *Multiuser resource allocation for mobile-edge computation offloading*, in Proc. IEEE Globecom, 2016.

[316] R. YU, Y. ZHANG, S. GJESSING, W. XIA, AND K. YANG, *Toward cloud-based vehicular networks with efficient resource management*, IEEE Network, 27 (2013), pp. 48–55.

[317] W. ZENG, Y. ZHAO, K. OU, AND W. SONG, *Research on cloud storage architecture and key technologies*, in Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human 2009, Seoul, Korea, 24-26 November 2009, 2009, pp. 1044–1048.

[318] J. ZHANG AND R. J. FIGUEIREDO, *Application classification through monitoring and learning of resource consumption patterns*, in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, IEEE, 2006, pp. 10–pp.

[319] J. ZHANG AND R. J. O. FIGUEIREDO, *Application classification through monitoring and learning of resource consumption patterns*, in 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece, 2006.

[320] J. ZHANG, X. HU, Z. NING, E. C.-H. NGAI, L. ZHOU, J. WEI, J. CHENG, AND B. HU, *Energy-latency trade-off for energy-aware offloading in mobile edge computing networks*, IEEE Internet of Things Journal, (2017).

[321] K. ZHANG, Y. MAO, S. LENG, Q. ZHAO, L. LI, X. PENG, L. PAN, S. MAHARJAN, AND Y. ZHANG, *Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks*, IEEE Access, 4 (2016), pp. 5896–5907.

[322] X. ZHANG, S. JEONG, A. KUNJITHAPATHAM, AND S. GIBBS, *Towards an elastic application model for augmenting computing capabilities of mobile platforms*, in Mobile Wireless Middleware, Operating Systems, and Applications - Third International Conference, Mobilware 2010, Chicago, IL, USA, June 30 - July 2, 2010. Revised Selected Papers, 2010, pp. 161–174.

[323] Y. ZHANG, G. HUANG, X. LIU, W. ZHANG, H. MEI, AND S. YANG, *Refactoring android java code for on-demand computation offloading*, in Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012, 2012, pp. 233–248.

[324] Y. ZHANG, G. HUANG, W. ZHANG, X. LIU, AND H. MEI, *Towards module-based automatic partitioning of java applications*, Frontiers of Computer Science, 6 (2012), pp. 725–740.

[325] Y. ZHANG, P. QU, J. CIHANG, AND W. ZHENG, *A cloud gaming system based on user-level virtualization and its resource scheduling*, IEEE Transactions on Parallel and Distributed Systems, 27 (2016), pp. 1239–1252.

[326] B. ZHAO, Z. XU, C. CHI, S. ZHU, AND G. CAO, *Mirroring smartphones for good: A feasibility study*, in Mobile and Ubiquitous Systems: Computing, Networking, and Services - 7th International ICST Conference, MobiQuitous 2010, Sydney, Australia, December 6-9, 2010, Revised Selected Papers, 2010, pp. 26–38.

[327] P. ZHAO, H. TIAN, C. QIN, AND G. NIE, *Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing*, IEEE Access, 5 (2017), pp. 11255–11268.

[328] W. ZHENG, P. XU, X. HUANG, AND N. WU, *Design a cloud storage platform for pervasive computing environments*, Cluster Computing, 13 (2010), pp. 141–151.

Résumé :

De nos jours, les terminaux tels que smartphones sont des dispositifs omniprésents, offrant des applications et des informations continuellement ‘‘à portée de notre main’’ allant de simples services de communications à des services multimédias. Les utilisateurs mobiles s'attendent à pouvoir utiliser des applications intensives en calcul, ce qui requière la disponibilité de ressources importantes, telles que jeux vidéo, reconnaissance faciale et vocale. Cependant, en raison des contraintes de légèreté, de maniabilité, de coût et de compacité, en plus de la sûreté et la mobilité, les terminaux sont nécessairement conçus avec des ressources contraintes, bien qu’en forte progression, qui incluent une capacité limitée de calcul CPU/GPU, une faible autonomie de batterie, et un espace mémoire restreint. Étant donné ces limitations, les applications exigeantes en ressources ne s’exécutent pas de la même manière sur ces terminaux en comparaison avec des architectures X86 (ex. PC de bureau ou serveur) et certaines applications ne peuvent tout simplement pas être utilisées sur ces terminaux. Pour pallier à ces problèmes, deux options s’offrent aux industriels et aux chercheurs :

- **Augmentation matérielle** : consiste à améliorer les capacités matérielles des terminaux, qui comportent les processeurs CPU et GPU, la batterie, et le stockage.
- **Augmentation logicielle :** consiste à utiliser les ressources d'infrastructure distante pour délester du calcul afin de conserver les ressources locales.

Plusieurs études ont été menées pour augmenter les performances des terminaux. Les fabricants proposent des processeurs multi-cœurs à fréquence d’horloge assez élevée. L'industrie ARM propose différents types de microprocesseurs qui répondent à des exigences de performance, de puissance et de coûts, tels que le *Cortex-A* qui intègre une unité de gestion de la mémoire (MMU), conçu pour exécuter des systèmes d'exploitation complexes, y compris Linux, Android et Microsoft Windows. Samsung et Qualcomm conçoivent de nouvelles générations de smartphones, qui intègrent des GPU, tels que *Mali* ou *Adreno*. Concernant l'autonomie de la batterie, de nombreux efforts ont été déployés pour récolter de l'énergie à travers le mouvement, l'énergie solaire et les radiations sans fil. Ces solutions sont encore en phase d'étude et ne peuvent malheureusement pas renouveler notablement l’énergie emmagasinée dans les batteries. En effet, ces ressources sont intermittentes et ne peuvent pas être disponibles à la demande. En parallèle à ces travaux, des chercheurs tentent de réduire la consommation énergétique dans différents aspects, y compris le matériel, les systèmes d'exploitation, applications et interfaces réseau. ‘‘Dynamic Voltage Scaling (DVS)’’ est une technique de gestion d’énergie, qui consiste à augmenter et diminuer la différence de potentiel au besoin ce qui permet de conserver l’énergie. L'arrêt automatique de l'écran est une autre alternative pour économiser de l'énergie sur les terminaux. Pour Samsung, la batterie amovible est un autre moyen pour augmenter l’autonomie des smartphone.

L'augmentation logicielle (ou ‘‘*computation offloading*’’ en anglais) a également attiré l’attention des industriels et chercheurs académiques. Cette solution qui conserve les ressources *rares* des terminaux en délestant des calculs vers des infrastructures ou serveurs distants. ‘‘Mobile Cloud Computing’’ (MCC) est envisagée comme une solution prometteuse pour relever les défis précédemment cités. Plusieurs approches ont été proposées dans le contexte du MCC; notamment le ‘‘*load sharing*’’ et l'exécution à distance qui ont évolué vers le concept de délestage de calcul ou ‘‘*computation offloading*’’, une approche plus générale et mature.

Le délestage de calcul consiste donc à exécuter à distance, dans un environnement Cloud avec des ressources suffisantes, une partie ou la totalité d’une application intensive en calcul. Cette technique

offre ainsi un moyen attrayant pour réduire le temps d'exécution exigé par les utilisateurs mobiles et/ou économiser la consommation énergétique sur leurs terminaux. Durant la période d'exécution à distance, soit le terminal continue à exécuter d'autres tâches, soit il attend les résultats d'exécution du serveur en charge de l'exécution à distance. À la fin de cette dernière, les résultats d'exécution de la partie d'application « migrée » sur le serveur sont utilisés par l'application exécutée sur le terminal. La computation offloading nécessite les quatre étapes suivantes:

1. Phase de profilage de l'application : consiste à disséquer l'application afin d'estimer la consommation en ressource CPU/GPU, batterie et mémoire. L'instrumentation du code est une des techniques utilisées pour mesurer les performances et connaître le comportement d'un programme. Elle consiste à insérer des instructions de mesure de performance à des endroits bien précis dans le programme dépendants de la granularité choisie qui varie d'un module (i.e. un ensemble de classes) à une fonction ou un bloc d'instructions dans une classe. Cette granularité est déterminée en fonction des performances et de la sécurité souhaitées. En effet, plus la granularité est large (classe ou module), plus la consommation est importante et plus le code source à migrer est compréhensible par des tiers (pirates par exemple).

2. Phase de modélisation de l'application : généralement, dans cette étape, l'application est représentée sous forme d'un modèle mathématique utilisant la théorie des graphes, la programmation linéaire, la théorie des jeux ou encore les processus de décision Markovien. La représentation tient compte de la granularité définie précédemment. Un exemple de représentation pourrait être un graphe pondéré où les sommets du graphe représentent les composantes de l'application (ex., fonctions, classes ou modules selon la granularité choisie) et les arcs décrivent les interactions entre ces composantes. Les sommets (arcs, respectivement) sont valués avec les performances obtenues dans la phase de profilage comme la consommation en CPU/GPU, mémoire, énergie (nombre d'interaction, type et taille des données échangées entre deux sommets, respectivement).

3. Phase de partitionnement de l'application : à ce niveau, une décision est prise pour chaque composante logicielle de l'application, à savoir si elle doit être déportée sur un serveur ou exécutée en local sur le terminal. La décision est basée sur la résolution d'une fonction objectif dont les critères consistent par exemple à améliorer les performances (ou le temps de réponse), réduire la consommation énergétique sur le terminal ou encore combiner deux ou plusieurs paramètres. Plusieurs algorithmes sont proposés pour résoudre la fonction objectif. Nous citons les algorithmes de découpage de graphes min-cut/max-flow comme l'algorithme de *pre-flow push* ou *Stoer-Wagner*. Le but de ces algorithmes est de partitionner le graphe en deux partitions selon une coupe minimale. Toutes les composantes qui seront exécutées sur le terminal seront regroupées dans la même partition locale, tandis que les composantes qui maximisent le gain, en étant exécutées à distance, seront rassemblées dans la partition distante. Le partitionnement de graphes est connu pour être NP-complexe, donc des heuristiques (ex. l'algorithme *branch and bound*, les algorithmes gloutons ou les algorithmes génétiques) sont proposées pour remplacer ces algorithmes min-cut/max-flow.

4. Phase de communication entre partitions : cette étape consiste à établir un lien de communication entre le terminal qui représente en général le client, et le serveur distant. Ce lien sera utilisé pour des communications entre le client et le serveur pour l'échange des données, de code, et de résultats. Une variété de solutions existe pour supporter des communications dans un contexte client-serveur, nous distinguons les ‘‘Remote Procedure Calls (RPC)’’, ‘‘Remote Method Invocations (RMI)’’ , streaming, et l'utilisation de proxy.

Selon que ces étapes soient réalisées au moment de l'exécution de l'application ou avant, nous distinguons trois approches de délestage de calcul :

a. Délestage statique : il se produit avant le démarrage de l'application. Dans ce type de délestage, l'application est partitionnée seulement une fois en deux partitions : la partition locale qui contient les composantes logicielles qui doivent s'exécuter localement sur le terminal et la partition distante qui englobera les composantes à exécuter à distance. L'application est ainsi partitionnée avant son exécution, durant les phases de développement ou d'installation.
b. Délestage dynamique : pour pallier aux limitations de l'approche statique (en particulier l'hétérogénéité de l'environnement), l'approche dynamique est proposée. Cette approche s'adapte mieux aux variations de l'environnement (ex. latence, qualité du réseau, bande passante, énergie restante sur le terminal, sa charge de travail et celle du serveur) durant l'exécution. Les différentes étapes de délestage sont effectuées durant l'exécution de l'application.
c. Délestage hybride : pour tirer profit des deux approches (statique et dynamique), l'approche hybride ou semi-dynamique est proposée. Dans cette approche, une partie de la décision est prise durant la phase de design de l'application par le programmeur ou durant la phase d'installation par des outils d'analyse statiques. L'autre partie est partitionnée durant la phase d'exécution de l'application (délestage dynamique).

Le délestage de calcul implique diverses parties prenantes: le terminal, le réseau, l'application, le serveur distant, l'utilisateur mobile, et le framework de délestage. En conséquence, plusieurs défis et problèmes sont apparus traitant du temps de réponse, de la consommation en énergie, de la sécurité, et des communications réseau. Différents travaux de recherche sont menés pour soulever ces verrous. Certains sont résolus, tels que la latence du réseau, en utilisant le ''Multi-access Edge Computing (MEC)'' ou le ''Fog Computing'' dans des solutions de délestage de calcul. En termes de communications réseau, des technologies d'accès multi-radio (multi-RAT) et une utilisation de la bande passante à la demande pour améliorer le débit sont proposées. Certains autres défis restent à résoudre, tels que la latence du Framework, son architecture, et les algorithmes que ce dernier embarque pour les prises de décision. Nos contributions couvrent certaines de ces questions.

1.1 Contributions de la thèse

La thématique générale de la thèse traite du délestage de calcul pour des applications temps réel, à forte demandes en ressources, et grande complexité, comme les moteurs de jeux 3D et la reconnaissance faciale. Les travaux de recherche abordent des sujets relatifs à la faisabilité, la performance, et la latence réseau impactant les solutions de délestage. Les contributions de la thèse sont triples:

1. ***Les moteurs de jeux:*** sous cet axe, il y a deux contributions [190, 192], divisées en trois parties :
   a. *Stratégie de déploiement:* les moteurs de jeux sont étudiés dans un contexte Cloud en termes de variabilité des demandes en ressources. Différents types de jeux sont testés, et les performances obtenues sont considérées pour la consolidation du serveur. Les fournisseurs de jeux dans le Cloud devraient trouver un compromis entre la qualité de l'expérience (QoE) de l'utilisateur et le nombre de moteurs de jeux hébergés par un serveur. Actuellement, les serveurs dans le Cloud sont répartis entre plusieurs moteurs de jeux en utilisant les technologies de virtualisation. Plus la demande en ressources d'un moteur de jeu est élevée, plus importantes seront les ressources à réserver à la machine virtuelle hébergeant ce moteur de jeu, ce qui entraîne une sous-exploitation du serveur. Si les demandes de ressources d'un moteur de jeu ne varient pas beaucoup dans le temps,

le fournisseur peut facilement prévoir cette demande et, par conséquent, consolider efficacement le serveur. Cependant, si la demande en ressource du moteur du jeu varie beaucoup, dans ce cas le fournisseur d'infrastructure Cloud devrait réserver des ressources pour accommoder les pics de consommation, qui est une perte de ressources car les pics surviennent rarement. Dans cette logique, le fournisseur d'infrastructure Cloud devrait trouver un compromis entre une forte consolidation et une QoE élevée, en utilisant l'étude proposée dans [190].

b. *Dissection du moteur de jeu:* les moteurs de jeux, à notre connaissance, ne sont pas étudiés du point de vue d'architecture interne. Les moteurs de jeux sont considérés comme des boîtes noires. En comparaison avec des boxes, telles que la XBox One, les terminaux smartphones et tablettes ne permettent pas d'exécuter des jeux 3D avec une qualité d'encodage assez élevée. Des solutions exploitant une infrastructure à distance, telle que le ''Cloud gaming'' et le délestage de calcul, représentent la prochaine étape vers l'amélioration de l'expérience des jeux. Par conséquent, disséquer, tester, et analyser le comportement d'un moteur de jeu est un pas vers une meilleure compréhension sur la façon de distribuer les moteurs de jeux sur réseau. Cette contribution vise à analyser le comportement des différents types de jeux, tester leur performance, identifier les goulots d'étranglement des ressources, et, surtout, extraire les ''call flows'' internes au jeu.

c. *Faisabilité de délestage de calcul*: Un moteur de jeu est composé d'un ensemble de modules comme le module de physique, de rendu, d'intelligence artificielle, d'animations, etc. Pour chaque module, nous représentons son architecture interne et les interactions entre les blocs composants le module. Cette manière de faire nous facilite l'étude de faisabilité de délestage de calcul pour les jeux vidéo. L'architecture du module de rendu est assez complexe, ce qui rend ardue l'implémentation du délestage de calcul pour ce module. En effet, à travers l'architecture de ce module, les blocs sont rarement indépendants, et les appels sont assez fréquents entre ces blocs générant un besoin important de communications entre le terminal et le serveur, ce qui limite le gain atteignable en termes de performances. Le module de physique peut être réparti en trois sous-familles qui communiquent à travers une classe. Il serait donc relativement simple de distribuer le calcul de ces trois sous-familles sur des entités distinctes. Le module de scripts est le seul module qui interagit directement avec le module de physique. Il serait donc logique de gérer cette famille de modules sur le même ordinateur que les modules de physique. Le module d'audio n'a aucune interaction avec les autres modules principaux et n'interagit qu'avec le thread principal du jeu. Ce module peut ainsi être migré et représenté en tant qu'interface (API) sur une machine distante. La communication entre le client et la machine distante peut se faire soit par des appels à distance comme les RPC ou en streaming. Le Module de l'intelligence artificielle peut être très intensif en calcul. Il est difficile, voire impossible, de simuler le jeu sur des dispositifs à ressources contraintes sans réduire la complexité de ce module. De plus, vu que ce module est, la plupart du temps, indépendant des autres, il peut être déporté dans son ensemble.

2. Délestage de calcul orienté moteur de jeu: sous cet axe, il y a deux contributions [188, 190]. En [190], un le délestage de calcul statique des modules d'un moteur de jeu est proposé. La méthode consiste à décomposer la scène d'un jeu en plusieurs patterns ou objets ; par exemple, l'avatar du joueur, l'avatar de l'ennemi, et l'environnement. Différents critères sont définis, à savoir la consommation en ressources, les contraintes de dépendance au code, la latence réseau et la bande passante. Selon ces critères, chaque objet du jeu est placé soit sur le terminal soit sur le serveur

distant. Dans [188], un délestage dynamique des modules du moteur de jeu a été proposé. Cette contribution est une amélioration de notre travail effectué dans [190], où nous avons proposé une heuristique pour planifier le placement des modules en utilisant les mêmes critères.

3. Délestage de calcul orienté MEC: nous avons une contribution sous cet axe [189]. Une des contraintes les plus pertinentes dans une solution de délestage de calcul est la latence réseau entre le terminal et le serveur. En effet, une latence importante limite l'applicabilité des solutions de délestage de calcul aux applications tolérantes aux délais. Pour les applications temps réel (i.e., sensible aux délais) telles que les jeux vidéo, une latence élevée est intolérable et dégrade la qualité perçue par les utilisateurs. Heureusement, avec le MEC et la transition vers la 5G, la latence du réseau sera considérablement réduite. Notre contribution consiste à tirer parti de l'architecture MEC pour prendre en charge le délestage de calcul. Pour cela, nous concevons un Framework chargé de l'orchestration du processus de délestage. Ce Framework est conçu avec trois entités : (i) une application ‘‘mobile edge’’ installée sur le MEC host dont le rôle est d'accéder aux API RNIS (Radio Network Information Service) de l'eNodeB et de récupérer des informations liées aux utilisateurs actifs dans la cellule, faire des approximations de latence réseau, et enfin décider en fonction de différentes informations si un utilisateur donné est autorisé à effectuer un délestage de calcul ou non ; (ii) le terminal qui hébergera les fonctionnalités de délestage, on parle ici de profilage, modélisation, partitionnement, et communication ; enfin (iii) le serveur localisé en bordure de réseau d'accès (« edge »), qui aura pour rôle d'exécuter le code délesté depuis le terminal, et lui renvoyer les résultats.

1.2 Organisation du manuscrit

Ce manuscrit est organisé en six chapitres, comme suit :

Le chapitre 2 introduit le lecteur aux concepts de base du délestage de calcul (ou ‘‘computation offloading’’) nécessaires à la compréhension des contributions de la thèse. Premièrement, nous identifions les étapes requises pour effectuer un délestage de calcul citées précédemment, à savoir ; le profilage de l'application, sa modélisation mathématique, le partitionnement du modèle, puis la communication entre partitions. Ces étapes clés du délestage, peuvent être faites durant l'exécution de l'application, ce que nous appelons ‘‘délestage dynamique’’ ou avant l'exécution, ce qui est connu comme ‘‘délestage statique’’. Deuxièmement, nous rappelons l'historique du mécanisme de délestage de calcul et ses évolutions. Nous terminons ce chapitre en présentant les avantages et inconvénients du délestage de calcul.

Le chapitre 3 présente, de manière exhaustive et qualitative, l'état de l'art du délestage de calcul pour compléter la compréhension du chapitre précédent. Ce chapitre est donc organisé selon une classification des travaux de recherche en fonction des différents acteurs impliqués dans le processus de délestage de calcul. Ces classes de taxonomie tournent autour de : l'utilisateur, le terminal, le réseau, l'application candidate au délestage de calcul, le serveur, et enfin le framework de délestage.

Le chapitre 4 aborde un type d'application temps-réel, à savoir les jeux vidéo. Dans un premier lieu, une dissection des moteurs de jeux est nécessaire à la compréhension de la contribution. De ce fait, le paysage des jeux vidéo et des moteurs de jeux en terme d'architecture, modules, types de jeux et contraintes est étudié et synthétisé dans ce chapitre. Dans un second temps, l'étape de profilage est mise en avant en réalisant une batterie de tests de performances sur neuf types de jeux différents, dans des environnements fixes et mobiles. La consommation en ressource CPU et GPU par frame et par module est utilisée pour définir une stratégie de déploiement du jeu soit sur une architecture traditionnelle type client/serveur, soit une architecture Cloud gaming, ou encore une architecture de délestage de calcul. La stratégie de déploiement est basée sur la variabilité des demandes en

ressource CPU/GPU par frame et la jouabilité du jeu. Suite à cela, nous nous sommes intéressés à la faisabilité de l'architecture délestage de calcul pour les jeux vidéo en présentant son apport en termes de ressources CPU/GPU. La dernière partie de ce chapitre traite du délestage statique des jeux et compare les résultats obtenus à ceux de l'architecture client/serveur et Cloud gaming.

Le chapitre 5 décrit une architecture de délestage de calcul pour les jeux vidéo. Nous commençons ce chapitre en passant en revue les études antérieures de délestage de calcul orienté moteurs de jeux. Ces travaux de recherche souffrent principalement de la non considération des jeux type 3D ''First Person Shooter (FPS)''. En effet, dans ces travaux, seuls des jeux 2D non-FPS, non complexes, et non gourmands en ressource sont considérés. Les jeux 3D sont plus complexes, demandent beaucoup de ressources CPU/GPU et, plus important encore, les jeux FPS sont des jeux temps réel où les délais d'interaction entre le joueur et sa machine doivent être très courts. Notre approche apporte une solution à ces problèmes. En premier lieu, nous décrivons la représentation d'une scène 3D et les objets/patterns qui y sont utilisés. Ensuite, nous présentons notre solution de délestage de calculs pour les jeux vidéo. Nous proposons une heuristique basée sur les communications réseau pour sélectionner les objets avec les modules associés qui devraient être déportés sur le serveur. Cette approche est testée ensuite, avec la mise en place d'un banc de tests composé d'un smartphone et d'un serveur. Les résultats en termes de consommation CPU/GPU et communications réseau sont présentés montrant un grand potentiel vers une amélioration des performances et la qualité d'expérience utilisateur.

Le chapitre 6 se concentre principalement sur le MEC. Nous avons vu à travers les chapitres précédents que la latence est primordiale dans les solutions de délestage de calcul, en particulier pour les applications temps réel. Pour éviter cette contrainte, nous proposons de rapprocher les serveurs qui doivent heberger les moteurs de jeux ou tout type d'applications candidate au délestage de calcul, de l'utilisateur final. En particulier, nous utilisons le MEC pour effectuer le délestage de calcul. Ce chapitre commence donc par présenter le MEC, son architecture, et décrivant les termes techniques utilisés à travers ce chapitre, en particulier ceux liés au MEC et aux réseaux LTE. Ensuite, nous passons en revue l'état de l'art des solutions de délestage orientées MEC. Suite à cela, nous proposons un framework qui s'appuie sur le MEC, ses services et API pour aider dans la prise de décision. Le framework est composé de trois entités: deux middlewares, l'un sur le terminal et l'autre sur le serveur MEC, et une application ''mobile edge'', responsable de la prise de décision de délestage de calcul. Le framework est ensuite testé dans un cas d'utilisation de reconnaissance faciale.

Enfin, le chapitre 7 traite de la conclusion finale, il synthétise les résultats obtenus à travers la thèse et présente des perspectives d'évolution concernant la sécurité, la mobilité, le placement des tâches, et enfin une description d'un modèle de ''Computation Offloading as a Service (COFFaaS)'' dans un contexte 5G.