From Mobile to Cloud: Using Bio-Inspired Algorithms for Collaborative Application Offloading

Roya Golchay



N° d'ordre NNT : 2016LYSEI009

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

préparée au sein de

l'INSA LYON

Ecole Doctorale N°512

Informatique et Mathématiques

Spécialité de doctorat : Informatique

Soutenue publiquement le 26/01/2016, par :

Roya Golchay

From Mobile to Cloud: Using Bio-Inspired Algorithms for Collaborative Application Offloading

---

Devant le jury composé de

Président :	Bernard Tourancheau,	Professeur,	Université Joseph Fourier
Rapporteurs :	Philippe Roose,	MCF HDR,	Université de Pau et des Pays de l'Adour
	Sophie Chabridon,	MCF HDR,	Télécom SudParis
Examinateur :	Philippe Lalanda,	Professeur,	Université Joseph Fourier
	Jean-Marc Pierson,	Professeur,	Université Paul Sabatier,Toulouse 3
Directeurs :	Frédéric Le Mouël,	MCF,	INSA de Lyon
	Stéphane Frénot,	Professeur,	INSA de Lyon

Thèse effectuée au sein du Centre d'Innovation en Télécommunications et Intégration de Services (CITI) de l'INSA de Lyon, équipe *Dynamic Software and Distributed Systems* (DynaMid)

Résumé

Actuellement les smartphones possèdent un grand éventail de fonctionnalités. Ces objets tout en un, sont constamment connectés. Il est l'appareil favori plébiscité par les utilisateurs, comme étant le plus efficace, pratique et nécessaire parmi tous les dispositifs de communication existants. Les applications actuelles développées pour les smartphones doivent donc faire face à une forte augmentation de la demande en termes de fonctionnalités - de la part des utilisateurs, en données collectées et enregistrées - de la part des objets IoT du voisinage, en ressources de calculs - pour l'analyse des données et le profilage des utilisateurs ; tandis que - dans un même temps - les smartphones doivent répondre à des critères de compacité et de conception qui les limitent en énergie et à un environnement d'exécution relativement pauvre en ressources. Utiliser un système riche en ressource est une solution classique introduite en informatique dans les nuages mobiles (Mobile Cloud Computing), celle-ci permet de contourner les limites des appareils mobiles en exécutant à distance, toutes ou certaines parties des applications dans ces environnements de nuage.

Cependant, l'exécution déportée (offloading) - mise en oeuvre dans des centres de données géographiquement éloignés - introduit une grande latence du réseau, qui n'est pas acceptable pour les utilisateurs de smartphone. De plus, une exécution déportée (offloading) massive sur une architecture centralisée, crée un goulot d'étranglement, qui empêche l'évolution requise par l'expansion du marché des dispositifs de l'Internet des choses. L'informatique brumeuse (libre traduction du Fog Computing) a été introduite pour ramener le stockage et la capacité de calcul dans le voisinage de l'utilisateur ou à proximité d'un emplacement précis. Certaines architectures émergent, mais peu d'algorithmes existent pour traiter les propriétés dynamiques de ces environnements.

Dans cette thèse, nous focalisons notre intérêt sur la conception d'ACOMMA (Ant-inspired Collaborative Offloading Middleware for Mobile Applications), un interlogiciel d'exécution déportée collaborative inspirée par le comportement des fourmis, pour les applications mobiles. C'est une architecture orientée service permettant de décharger dynamiquement des partitions d'applications, de manière simultanée, sur plusieurs clouds éloignés ou sur un cloud local créé spontanément, incluant les appareils du voisinage. Les principales contributions de cette thèse sont doubles. Si beaucoup d'intergiciels traitent un ou plusieurs défis relatifs à l'éxecution déportée, peu proposent une architecture ouverte basée sur des services qui serait facile à utiliser sur n'importe quel support mobile sans aucun exigence particulière. Parmi les principaux défis il y a les questions de quoi et quand décharger dans cet environnement changeant et très dynamique, où le profile et le contexte du dispositif mobile, et les propriétés du serveur jouent un rôle considérable dans l'efficacité. A cette fin, nous développons des algorithmes de prises de décisions bio-inspirées : un processus de prise de décision bi-objectif dynamique avec apprentissage et un processus de prise de décision en collaboration avec les autres dispositifs mobiles du voisinage.

Nous définissons un mécanisme de dépôt d'exécution avec une méthode de partitionnement grain fin de son graphe d'appel. Nous utilisons les algorithmes des colonies de fourmis pour optimiser bi-objectivement la consommation du CPU et le temps total d'exécution, en incluant la latence du réseau. Nous montrons que les algorithmes des fourmis sont plus facilement re-adaptables face aux modifications du contexte, peuvent être très efficaces en ajoutant des algorithmes de cache par comparaison de chaîne (string matching caching) et autorisent facilement la dissémination du profil de l'application afin de créer une exécution déportée collaborative dans le voisinage.

Abstract

Not bounded by time and place, and having now a wide range of capabilities, smartphones are all-in-one always connected devices - the favorite devices selected by users as the most effective, convenient and necessary communication tools. Current applications developed for smartphones have to face a growing demand in functionalities - from users, in data collecting and storage - from IoT device in vicinity, in computing resources - for data analysis and user profiling; while - at the same time - they have to fit into a compact and constrained design, limited energy savings, and a relatively resource-poor execution environment. Using resource- rich systems is the classic solution introduced in Mobile Cloud Computing to overcome these mobile device limitations by remotely executing all or part of applications to cloud environments. The technique is known as application offloading.

Offloading to a cloud - implemented as geographically-distant data center - however introduces a great network latency that is not acceptable to smartphone users. Hence, massive offloading to a centralized architecture creates a bottleneck that prevents scalability required by the expanding market of IoT devices. Fog Computing has been introduced to bring back the storage and computation capabilities in the user vicinity or close to a needed location. Some architectures are emerging, but few algorithms exist to deal with the dynamic properties of these environments.

In this thesis, we focus our interest on designing ACOMMA, an Ant-inspired Collaborative Offloading Middleware for Mobile Applications that allowing to dynamically offload application partitions - at the same time - to several remote clouds or to spontaneously-created local clouds including devices in the vicinity. The main contributions of this thesis are twofold. If many middlewares dealt with one or more of offloading challenges, few proposed an open architecture based on services which is easy to use for any mobile device without any special requirement. Among the main challenges are the issues of what and when to offload in a dynamically changing environment where mobile device profile, context, and server properties play a considerable role in effectiveness. To this end, we develop bio-inspired decision-making algorithms: a dynamic bi-objective decision-making process with learning, and a decision-making process in collaboration with other mobile devices in the vicinity. We define an offloading mechanism with a fine-grained method-level application partitioning on its call graph. We use ant colony algorithms to optimize bi-objectively the CPU consumption and the total execution time - including the network latency.

List of the acronyms

MC	Mobile Computing
CC	Cloud Computing
MCC	Mobile Cloud Computing
ACO	Ant Colony Optimization
IoT	Internet of Things
SPC	Spontaneous Proximity Cloud
DS	Distant Cloud
SM	String Matching
SP	Shortest Path
BSP	Bi-objective Shortest Path

Table 1: List of the acronyms

Acknowledgements

Getting a doctorate, undoubtedly, was a challenging process in my life. I would like to express my very great appreciation to all those who helped me by their guidelines and supports to successfully complete this course.

First, I would like to offer my special thanks to Stéphane for the opportunity he provided me to start PhD and also for his availability. I wish to acknowledge my supervisor, Frédéric, who trusted me, helped me grow as a researcher and gave me a new dimension to my way of thinking. I am grateful to all his personal and academic comments and his different attitudes to life. He always gave me confidence that I would be successful despite all obstacles and helped me get to the end.

My thanks also go to all Laboratory partners for the friendly atmosphere and great times we spent together. My special thanks are extended to Fabrice and Florent for their support and attention and also to Marie Ange because of her warm company and encouragement in the most difficult times.

Special thanks should be given to my good friends whose company energetized me to work hard, especially Marie whose kindness cannot be reciprocated with words.

I am deeply grateful to my parents for their unconditional love, companionship and support at all stages of my life. I wish to thank my Dad for reminding me of the importance of teaching and research and my Mom for her immeasurable support and assistance. I would like to express my deep gratitude to my brothers, Hamed and Behnood, for their encouragement and endless kindness. I would like to express my very great appreciation to my dear husband, Hamidreza, for all encouragement and conditions provided for me.

And finally, I wish to thank my lovely daughters, Pania and Parmis, who have filled every moment of my life with joy and indescribable sweetness. I would like to do apologize about all the hours and days they spent without Mom or with a busy and bored Mom.

It should be noted that this list is not exhaustive and I really acknowledge every people who contributed to my project.

Contents

1 Introduction **1**
- 1.1 Mobile Device: From Personal Device to IoT Gateway . . . 1
- 1.2 From Mobile to Cloud: Mobile Cloud Computing . . . 5
- 1.3 Problem Statement: Application Offloading . . . 9
- 1.4 Thesis Outline . . . 11

2 Overview of the Application Offloading Middleware **13**
- 2.1 General Architecture of Mobile Cloud Computing . . . 13
- 2.2 Architecture and Communication for Cloud Application Engineering . . . 15
  - 2.2.1 Client-Server Based Architecture . . . 16
  - 2.2.2 Virtualization Based Architecture . . . 20
  - 2.2.3 Mobile Agent Based Architecture . . . 23
- 2.3 Offloading Destination . . . 24
  - 2.3.1 Distant Cloud Based Middleware . . . 25
  - 2.3.2 Local Cloudlet Based Middleware . . . 29
  - 2.3.3 Proximate Cloud Based Middleware . . . 31
- 2.4 Middleware Classification Based on Decision Making Process . . . 33

3 An Automated Application Offloading Middleware **37**
- 3.1 Main Contributions of Designing Offloading Middleware . . . 38
- 3.2 An Overview of Mobile Applications from an Application Engineering Perspective . . . 40
  - 3.2.1 Mobile Application Architecture . . . 42
  - 3.2.2 Mobile Application Transformation . . . 44
- 3.3 An Overview of Application Offloading Middleware from a Runtime Perspective . . . 47
  - 3.3.1 Design Objectives . . . 47
  - 3.3.2 Service-oriented Architecture for ACOMMA . . . 50
    - 3.3.2.1 Service Description . . . 50
    - 3.3.2.2 Service Interactions . . . 52

4 Individual Offloading Decision Making in ACOMMA **55**
- 4.1 An Introduction of Decision Making Process for Application Offloading . . . 55
  - 4.1.1 Different Aspects of Offloading Decision Making . . . 56
  - 4.1.2 Application Partitioning Problem Considered as Shortest Path Problem . . . 60
  - 4.1.3 Solving Shortest Path Problem Using Bio-Inspired Algorithms . . . 63
- 4.2 Decision Making Process of ACOMMA for Application Offloading . . . 65

4.2.1 Application Offloading Flow of ACOMMA . . . 65
4.2.2 Bi-Objective Offloading Decision Making Using Ant Colony Optimization Algorithm . . . 67
4.2.3 Learning-Based Offloading Decision Making Using String Matching Algorithm 71

5 Collaborative Application Offloading 75
5.1 An Introduction of Collaboration-based Application Offloading . . . 75
5.2 Collaborative Offloading in ACOMMA . . . 79
5.2.1 Collaboration-Based Resource Sharing in Application Offloading . . . 80
5.2.1.1 Creating Service Graph for Multi Destination Application Offloading 82
5.2.1.2 Applying ACO for Multi Destination Decision Making . . . 84
5.2.2 Collaboration-Based Decision Sharing in Application Offloading . . . 86
5.2.2.1 Collaborative Decision Sharing . . . 86
5.2.2.2 Decision Cache Management . . . 87

6 Implementation and Evaluation of ACOMMA 91
6.1 Validation Approach . . . 91
6.1.1 Applications . . . 91
6.1.2 Experimental Platform . . . 93
6.1.3 Success Criteria . . . 94
6.2 Evaluation . . . 94
6.2.1 Evaluation of Individual Decision Making for Single Destination Offloading . 94
6.2.1.1 Ant Colony Optimization Performance . . . 95
6.2.1.2 String Matching Performance . . . 100
6.2.2 Evaluation of Collaborative Offloading . . . 101
6.2.2.1 Decision Sharing . . . 102
6.2.2.2 Resource Sharing . . . 104

7 Conclusion 107
7.1 Summary . . . 107
7.2 Short and Long Term Perspectives . . . 109

List of Figures

1.1 Application offloading dimensions 9

2.1 General architecture of Mobile Cloud Computing 15
2.2 Affecting factors in Mobile Cloud Computing 16
2.3 The architecture of Cuckoo [69] 19
2.4 The architecture of MAUI 21
2.5 The architecture of ThinkAir 26
2.6 Fuzzy logic system 28
2.7 VM synthesis 30
2.8 Thematic taxonomy of application offloading middlewares 34

3.1 Mobile devices as IoT gateway and SPC 38
3.2 A sample call graph 41
3.3 Mobile application architecture of ACOMMA from application engineering point of view 44
3.4 Mobile application execution flow with servicization modifications 46
3.5 A general execution flow of offloading middleware 49
3.6 Layered architecture of application offloading middleware 50
3.7 Service interactions of middleware architecture 53

4.1 Transforming call graph to be compatible to SP problem 61
4.2 A weighted call graph with local and remote paths 63
4.3 An architectural view of offloading building blocks in ACOMMA 66
4.4 Offloading process in ACOMMA 67
4.5 Non dominated solutions for Shortest Path Problem 69
4.6 Decision making process using String Matching 72
4.7 A sample of String Matching Cache 74

5.1 A motivating scenario to make mobile devices collaborate 77
5.2 Centralized client-server vs. decentralized peer-to peer communication 79
5.3 Building blocks of ACOMMA for collaborative offloading 81
5.4 General flow of collaborative offloading 82
5.5 Application partitioning for multi destination offloading 83
5.6 Call graph modification for multi destination offloading 84
5.7 A decision cache composed of decision trails and contextual information 87

6.1 Local and ACO offloading execution time of micro benchmarks on Galaxy SII 96

6.2 Local and ACO offloading execution time of micro benchmarks on Google Nexus 7 Tablette 97
6.3 Local and ACO offloading execution time of macro benchmarks on Galaxy SII . . . 98
6.4 ACOMMA's overhead running ACO for successful runs of Determinant and Integral 98
6.5 Local and ACO offloading CPU usage of micro benchmarks on Google Nexus 7 Tablette 99
6.6 Nearby device discovery 102
6.7 Execution time of application offloading, using SM by local or collaborative cache . 103

List of Tables

1 List of the acronyms v

2.1 A classification of application offloading middlewares 35

6.1 Test inputs for individual decision making using ACO 95
6.2 Summary of individual decision making using ACO on micro benchmarks 100
6.3 Summary of individual decision making using ACO on macro benchmarks 100
6.4 Execution time gained bye SM algorithm compared with ACO 101
6.5 Test inputs for collaborative decision sharing using ACO 102
6.6 Execution time gained by SM using collaborative cache compared with local cache . 103
6.7 Execution trace of Determinant for multi destination offloading 104
6.8 Execution trace of Integral for multi destination offloading 104

Chapter 1

Introduction

1.1 Mobile Device: From Personal Device to IoT Gateway 1
1.2 From Mobile to Cloud: Mobile Cloud Computing 5
1.3 Problem Statement: Application Offloading 9
1.4 Thesis Outline 11

The goal of this chapter is to highlight the challenges in computer system research raised by Mobile Cloud Computing and put the light on new challenging problems, the management of dynamic and scaling aspects in application offloading. We begin by examining the dual role of mobile device and specially smartphone as a personal device and as a gateway of Internet of Things. Then we mention the motivation of application offloading from mobile device to the cloud to overcome its inherent resource limitations. We especially focus on two hot key research points on Mobile Cloud Computing: dynamic decision making for offloading and collaboration between mobile devices and IoT from offloading point of view. Finally, we outline the contributions of this thesis and detail the chapter contents.

1.1 Mobile Device: From Personal Device to IoT Gateway

The user tends to use more and more smartphones instead of portable computing and communication devices as all-in-one always-connected devices with custom-built personal productivity, social media, and entertainment significantly increased so that becoming less of a luxury and more of a necessity in human life. According to CISCO Visual Networking Index [30] average smartphone usage grew 50 percent in 2013. By the end of 2014, the number of mobile-connected devices exceeded

the number of people on earth, and by 2019 there will be nearly 1.5 mobile devices per capita.

Ubiquitous network connectivity, even for these small devices in addition to their mobility property make users to rely on their mobile devices as their go-to devices and increasingly using them for daily tasks such as Internet banking, emailing, and emergencies (such as viewing online traffic map or using routing applications to find the best shopping way or connecting to a medical information system to take a prescription urgently). The expansion of mobile devices usage offers large number of features to users. Indeed, these platforms make services, applications and functionalities available anywhere and any time.

Wide range of smartphones capabilities typically including 3G connectivity, GPS, WiFi, high quality graphic, cameras,various sensors, gigabytes of storage and gigahertz speed processors beside their compact design and miniature nature which make them dominant computing devices selected by users highly motivate application market to develop wide range of resource constrained mobile applications such as m-commerce, mobile telemedicine, multiplayer mobile gamming, machine learning, natural language processing, pattern recognition, augmented reality service to satisfy high users expectation. However the inherent limitations of smartphones make it difficult to exploit their full potential to run these complex applications and have the best performance compared with the same application running on powerful stationary computing devices.

However not bounded by time and place make the mobile devices become an essential part of human life as the most effective and convenient communication tools but its more than just connecting people. Mobile is about connecting everything.

Machine-to-machine (M2M) mobile connectivity represents the next great wave of what mobile can make possible in our lives. Huawei [134] uses M2M also standing for Machine to Man, Man to Machine, or Machine to Mobile. As wireless innovation continues, this M2M connectivity the so-called "Internet of Things" intelligently connects humans, devices, and systems and will further accelerate mobile opportunity and transform how people and our economy interact with the many tools of modern life. Through mobile connectivity, the Internet can make virtually anything more intelligent: holding great promise for our economy, our environment, our education and health care systems, our safety and our standard of living. From a refrigerator and home thermostat to a car or office whiteboard to a child's textbook or doctor's medical tablet, wireless technology is leaping beyond the phone to connect the world around us to the Internet and this "Internet of Things"

will greatly improve our lives and our economy.

When embedded with chips and sensors, these objects(Things) can "think", "feel", and "talk" with each other. Together with the infrastructure of the Internet and mobile networks, these objects can communicate with humans, and enable us to monitor and control them anytime anywhere and enjoy their intelligent service, making the idea of a "Smart Planet" a dream come true.

Exponentially augmented number of connected devices to IoT make an old science fiction history comes true. According to Gartner[1] the IoT will reach 26 billion connected devices in 2020, with an exponential growth of 30 times the installed base in 2009, when connected devices in the web were just 900 million [90]. According to ABI Research[2] [89] more than 30 billion devices will be wirelessly connected to the IoT (Internet of Everything) by 2020.

The smartphones brimming with sensors (an accelerometer, a compass, GPS, light, sound, and altimeter) are becoming a universal interface and remote control for these things. In this way, the smartphone is a gateway drug for you to enter the next level, in which the internet is in your thermostat, lights, door locks, car and wristwatch. For instance, in order to reduce car accidents and provide drivers with an easier and safer way of manipulating the navigation system, [132] introduces a remote control framework that make remote person capable to control the car navigation system on behalf of the driver. Using flexible home control and monitoring system proposed in [98], any Android based smartphone with built in support for Wi-Fi can be used to access and control the devices at home. Another example is a thermostat system in a house that could be controlled via the personal IoT platform using smart phone [128].

Smart homes, Internet-connected cars, and wearable devices which represent the next generation of mobile gear beyond smartphones, are new systems that will coexist with phones for at least the next few years.

Despite increasing usage of smartphones either for individual owner use as mini-computers that travel with them and keep them connected 24 hours a day for running different mobile applications, or as the universal interface for IoT, exploiting their full potential is difficult due to their inherent limitations. This often severely constrains hardware and software development for these devices. [103] divides constraints raised by mobile computing into the three main categories of

[1]American information technology research and advisory firm headquartered in Stamford, Connecticut, United States

[2]Market research and market intelligence firm based in New York

mobile device, network and mobility constraints. We do the same division but we consider mobile device as a gateway for the IoT as well as ordinary personal device.

- Mobile Device Constraints: Due to their small size and weight mobile devices are resource-poor in terms of processor speed, storage space, display size and screen resolution compared with stationary computers. This is what is mentioned in almost all researches done in Mobile Computing domain. We believe that although mobile devices are not as powerful as stationary computers, by the virtue of rapid and drastic progress in embedded technologies they are strong enough to meet user requirements itself and the user rather suffers from short battery life time. This battery shortage is more problematic while connecting to IoT as a gateway. In addition, hardware resources of mobile device could not support new volume of processing imposed by its new role. How to meet these constraints plus power limitation while satisfying user is a great challenging point for developers [61].

- Network Constraints: Because of their mobility nature, mobile devices use wireless networks instead of wired ones. Despite great improvement in wireless network they will still continue to have limited bandwidth, high latency and frequent disconnections due to power limitations, available spectrum and mobility.

- Mobility Constraints:

  - Mobility is inherently hazardous [108]: While mobile devices are in move they are more vulnerable to loss or misplace, damage or theft. Their mobility makes it difficult to consider their availability. Security and privacy points are also much more important than stationary computer. Mobile devices record various private data about user same as location and this sensitive information should not be accessible to others without owner authorisation.

  - Mobile connectivity is highly variable in performance and reliability: Wireless network coverage is varying in different geographical position. There maybe no network coverage in some places. In addition network providers offer different bandwidth and connection speed. Some buildings may offer reliable, high-bandwidth wireless connectivity while others may only offer low-bandwidth connectivity. Outdoors, a mobile element may surely have to rely on a low-bandwidth wireless network with gaps in coverage. If

the system does not recognize and adapt to these differences, it can impact the user experience. For example, if a system sends high-quality video to a device with a very limited wireless connection, the result is long loading times and a poor user experience.

– Mobile device is not accessible for a specific period of time: Mobility makes it difficult to rely on a mobile device as an IoT gateway. At any movement, a mobile device should rediscover its environment to know nearby mobile devices as well as IoT. In addition, the availability of mobile device up to the end of IoT related processes is not guaranteed because of its mobility.

Great and consecutive improvement in mobile devices and network communication is required in order to overcome challenges raised by the aforementioned constraints. Augmentations in mobile device side divided into hardware and software approaches. Hardware approaches focus to empower mobile devices by exploiting powerful resources and long lasting battery.

Hardware solutions are not always feasible; generating powerful device caused additional heat, size and weight, preparing last longing battery in small device with enlarged resources is not possible with current technologies and finally equipping mobile device with high-end hardware will noticeably increasing the price [14]. On the other hand, software development process is much more faster than hardware development. There is a highly impressive development in software domain in the last decades.

This is why we are interested in software level solutions to atone hardware limitation in mobile computing. In the next section we introduce the most recent software approach named Mobile Cloud Computing after presenting a brief history of software approaches and its categories.

1.2 From Mobile to Cloud: Mobile Cloud Computing

Empowering mobile devices using software solutions is not a new concept. Different approaches including load sharing [95], remote execution [106], cyber foraging [109], and computation offloading [75], [76] try to improve performance and energy consumption of resource-poor mobile devices by using the power of one or more resource-rich stations. Due to slight differences among their concepts, researchers use the terms “remote execution”, “cyber foraging” and “computation offloading” interchangeably in the literature with similar principle and notion [14]. “Offloading” is

the term which we use in the rest of this dissertation for this concept.

The basic idea of code offloading is the core concept of different researches over the years [93, 18, 21, 19] as one of the most practical solutions to alleviate resource limitation in smartphones. A key area of application offloading is to make a resource-intensive device use remote execution to improve performance and energy consumption. The surrogate can be a powerful stationary device or a set of processors. Drastic evolution of wireless technologies that make network connectivity ubiquitous and successful practices of Cloud Computing for stationary machines are motivating factors to bring the cloud to the vicinity of a mobile from an offloading perspective. As result Mobile Cloud Computing was introduced to enable rich mobile computing by extending the on demand computing vision of CC and enrich smartphones and address their issues of computational power and battery lifetime by executing complete mobile applications or identified resource intensive components of a partitioned mobile application on cloud-based surrogates.

Although the offloading method is not a new concept, the term Mobile Cloud Computing was introduced and used about the same time as generalization of CC. Then in 2010, Google CEO, Eric Schmidt, explained MCC in an interview. Increasing use of mobile devices, especially smartphones, on the one hand and the many benefits of using MCC on the other hand have attracted a lot of attention to this new concept and prompted much research done in this area. The advantages of MCC can be divided as follows:

- Strengthening the processing power: Sending all or part of computation intensive mobile application to a reliable and strong resource can increase the prepossessing power and available memory while reducing execution time.

- Prolonging the battery: Offloading process on a cloud-based device considerably decreases energy consumption and increases mobile device battery life during the execution of the energy-intensive application.

- Unlimited storage: Being connected to cloud and its almost unlimited storage compared with the limited storage of mobile device results in an increase in the available storage capacity in mobile device.

- Data Safety: By storing the sensitive data on secure and reliable resources of a cloud, it reduces the possibility of these data being stolen, lost or physically damaged.

- Data sharing and ubiquitous access: Users of mobile devices will be able to access their stored data on the cloud storage anytime, anywhere and through any device.
- Enriched user interface: Due to the inherent limitations of mobile devices, heavy and compact 2D and 3D screen rendering can be done in the cloud and the final image will be prepared based on the features and screen size of the mobile device.
- Enhanced Application Generation: Distributed Mobile Applications can be implemented for a variety of dissimilar mobile devices by using already developed components on the clouds.

The main objective of MCC is to exploit the above-mentioned benefits and provide a method for fast and easy access to cloud resources. Naturally, there are some problems in achieving these goals. Since MCC has arisen from a merger of Mobile Computing and CC, important factors that affect the quality of MCC are challenges related to MC and CC and the relationship between these two:

- Cloud side challenges
  - Privacy: Although the protected state of data stored in the cloud avoids them being lost or destroyed, the public part of cloud space can compromise the users privacy despite the creation of separate virtual space for each user.
  - Security: Protecting the security of the data during transmission to the cloud and vice versa and during their residence on cloud is a noteworthy point in discussions on CC and consequently in MCC.
  - Cost: cloud is a non-free infrastructure and the user must meet its cost to be able to use it. Estimating whether paying such a cost is effective for the user and when the user is willing to pay it are other issues to be considered.
- Communication challenges
  - Communication Protocol: Although communication protocols such as Hadoop [9] which is implemented in distributed computing as well as open resource APIs of the cloud itself like Dropbox [2], Azur [10] and OpenNebula [12] can be used in MCC, the lack of a standard communication protocol especial(customized) for MCC is one of its drawbacks that must be examined more.

  - Infrastructure deployment: As an IoT gateway, a mobile device needs to communicate with heterogeneous sensors. These sensors usually use private communication protocols in their network that are known only within their network. How to deploy an infrastructure to communicate with these sensors on various communication networks while having reasonable response time and how to connect them to the cloud is a challenging point.

- Mobile side challenges

  - Complexity of application implementation: Applications that can be transferred into a cloud are more complex than normal applications and their implementation requires more knowledge, skills and time. When some parts of the application need to be transferred to the cloud, the developer is responsible for detecting and annotating the portable parts.
  - Performance: Although the main goal of MCC is to overcome the problems caused by restrictions of mobile device resources through offloading the entire application or costly parts of it in order to increase efficiency, communication with cloud and sending and receiving information require large amounts of resources and is costly. How to make an efficient offloading decision to augment mobile device performance concerning offloading costs and benefits is a challenging point while designing an offloading middleware.
  - Mobility transparency and awareness: Mobility of mobile device itself as well as its dynamic and highly changing environment also cause some challenges. Mobility may interrupt mobile devices' connection with the cloud during the execution of some parts of the application on the cloud, therefore, it can lead to the impossibility of mobile devices' access to the processing results. Making these issues transparent to mobile device is important. On the other hand, mobile devices specially used as IoT gateway need to be aware of their highly changing environment to be able to communicate with cloud as well as IoT. This awareness may lead to making more efficient offloading decisions. Being mobility transparent and mobility aware at the same time depending on situation is an important challenging point while offloading. Making users to agree to collaborate with others using incentive methods is challenging. How to make this collaboration be also beneficial for mobile device itself is a point of discussion.

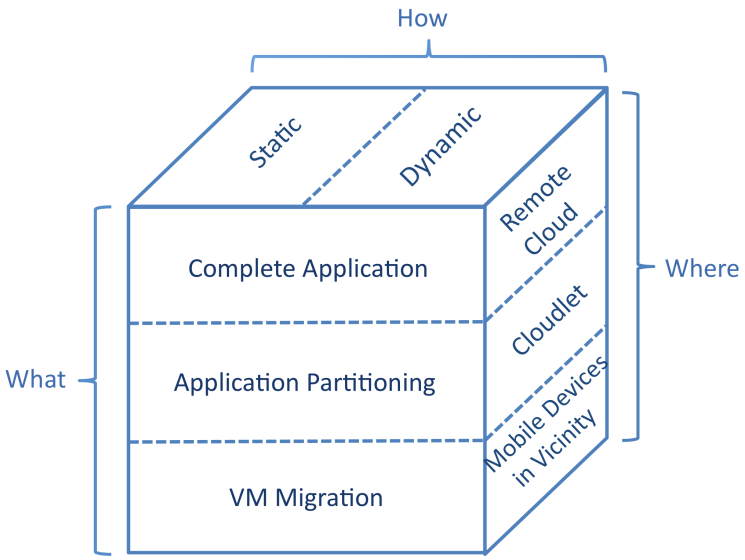

Figure 1.1: Application offloading dimensions

In this thesis we focus on challenging points related to mobile device and introduce our offloading middleware as a response to these issues.

1.3 Problem Statement: Application Offloading

How to make mobile devices benefit from CC using application offloading is an important research point in MCC. There are different ways to delegate resource / computation-intensive parts of a mobile application to more powerful machine that their choices based on the context and requirements may result in different performance. Answering some key questions as follows leads to designing an offloading middleware that meets performance goals in an efficient way.

- What to offload?
- Where to offload?
- How to offload?

Figure 1.1 shows a summary of existing answers to these questions.

The proposed offloading solutions are generally to develop an architecture in which the mobile is charged with the responsibility of defining what should be offloaded based on two main strategies: entire application offloading and application partitioning. The virtual machine techniques used

to migrate the entire application process in first offloading form where code partition becomes transparent to programmer [110], [28] while in application partitioning form the application is partitioned either statically using basic implementation primitives (e.g. annotations) or dynamically using component-based application partitioning techniques [84] and part of the code is outsourced based on available resources, such as network availability, bandwidth and latency [32], [91], [55]. These portions could have different granularity same as OSGi bundles [32] or methods [73].

There are three main surrogate types for mobile device to offload the above mentioned parts. The first one is the real cloud resources with static hardware infrastructure. Googles Gmail for Mobile [8] is an example which uses rich Google servers while the mobile device uses 3G connection to communicate with a remote server as a thin client. Facebooks' location aware services [5] and Twitter for mobile [13] are some other example of this type. The second surrogate type is a closer network layer to mobile device called "Cloudlet" that was proposed in [110] and described as "data center in a box". Cloudlet is a set of several multi-core computers with connectivity to the remote cloud servers. Mobile devices in local vicinity can also be considered as resource providers with peer-to-peer communication. Other available stationary devices can be used in addition to these collective devices of various mobile devices. Hyrax [83] is an example which uses this surrogate type while offloading.

The total process of offloading can be done either statically while implementing or dynamically at run time. In the first case, offloadable parts are defined at the beginning of an application and they will not change during the execution. In the second case, the decision making engine will decide for offload able portions based on the current situation at runtime.

To be adapted to the mobility of the mobile device itself and its environment while offloading, we make a contribution on an offloading middleware with an open architecture which makes dynamic offloading decisions at runtime considering the current situation of a mobile device and its context. To perform an adaptable and scalable offloading, we apply fine-grain method level offloading. We propose a bio-inspired algorithm to take offloading decisions in a dynamic way.

To benefit form physical proximity of an IoT to the mobile device compared with Cloudlet and DS, we consider nearby devices as a cloud that constructs and destroys itself spontaneously. Then we propose an approach to make these nearby devices to collaborate in the event of offloading. The fine-granularity of offloadable parts makes it possible to execute on small nearby mobile devices.

1.4 Thesis Outline

In this the first chapter, we presented the concept of offloading in MCC and then we highlighted new challenging points in the domain considering the dual role of a mobile device in its dynamic environment as a personal device and as an IoT gateway at the same time. Finally, we shortly mentioned how we address these issues. This manuscript is organized as follows:

- **Chapter 2** provides a survey of existing application offloading middleware and brings out drawbacks and limitations of currently existing solutions. A classification of these approaches under new key research points is presented at the end of the section.
- **Chapter 3** firstly exposes our main contributions of designing an offloading middleware. It then describes how to make a mobile application be adapted to our offloading middleware. Descriptions of service-based and open architecture of this middleware as well as a description of the services and their interactions can be found in the last section of this chapter.
- **Chapter 4** presents the decision making process for application offloading and explains our proposed bio-inspired algorithm to make a dynamic offloading decision. Learning-based offloading decision making using past decisions and its string matching algorithm is also illustrated in this chapter.
- **Chapter 5** describes our second contribution: making a collaborative offloading in cooperation between nearby mobile devices. Decision sharing and resource sharing are introduced as two different interests for making mobile devices to collaborate for offloading. How our middleware operates to offload onto nearby devices or benefits from others' decisions to make its own is explained in this chapter.
- **Chapter 6** is in fact a proof of concept. It explains our methodology for implementing our offloading middleware. Micro and macro benchmarks, different test scenarios and the results of several tests are presented in this chapter to show the performance of our proposed middeware under different circumstances.
- **Chapter 7** concludes and provides a summary of this work and presents the major perspectives of this work.

Chapter 2

Overview of the Application Offloading Middleware

2.1 General Architecture of Mobile Cloud Computing 13
2.2 Architecture and Communication for Cloud Application Engineering 15
2.2.1 Client-Server Based Architecture 16
2.2.2 Virtualization Based Architecture 20
2.2.3 Mobile Agent Based Architecture 23
2.3 Offloading Destination 24
2.3.1 Distant Cloud Based Middleware 25
2.3.2 Local Cloudlet Based Middleware 29
2.3.3 Proximate Cloud Based Middleware 31
2.4 Middleware Classification Based on Decision Making Process 33

To benefit from the advantages of MCC and to meet its challenges cited in Chapter 1, many improvements are made regularly on design and development of application offloading middlewares. In this chapter we prepare a state of the art of existing approaches considering the effects of application nature, from application engineering perspective, as well as cloud type on middleware design and offloading performance. We propose a thematic taxonomy of existing approaches and also a classification of them.

2.1 General Architecture of Mobile Cloud Computing

Despite slight differences in dThifferent definitions of MCC which is sometimes referred to as the future of mobile applications [101], offloading has been regarded to be the core of MCC in all defi-

nitions. Therefore a lot of methods have been recently proposed in this regard for the code, computation or application offloading. Furthermore, there have recently been many programs in different contexts which support CC. Among such programs, it can be referred to commerce [131],healthcare [43], [117],education [48], [133], social networks [4], file sharing [6], and searching [96] applications.

Various surveys have also tried to highlight the importance of MCC and offloading from different points of view [107], [80], [114], [111], [37]. The authors of [104], [86], [120], [111] examined the challenges and problems in MCC and [100] gave also a perspective. [69], [33], [126] focused on MCC applications. [33] presented a basic comparison of MCC applications and classified them and in [69] mobile cloud application models and their strengths and weaknesses as well as the parameters that influenced them were also reviewed. Unlike other surveys, [74] classified the methods of computation offloading based on the year they were found. The authors of [14] did one of the most complete surveys in this domain. In addition to providing a taxonomy of offloading methods that is called Cloud based Mobile Augmentation, they evaluated different types of remote resources and their impact on the quality of offloading. Items related to decision making, factors affecting efficiency and existing challenges are among the other topics discussed in this survey. Offloading frameworks entitled Distributed Application Processing Frameworks were explained and classified in [114] and the challenges and issues of their development and implementation in MCC is also highlighted.

The figure 2.1 shows the overall architecture of MCC. Mobile devices ranging from smartphone, tablet, PDA, etc can be connected through the infrastructure network to the cloud. Depending on the local conditions of a mobile device, this connection can be established via satellite, access point or BTS with the help of wifi / 3G / wi-max and LTE technologies. Amazon Elastic Compute Cloud (EC2) [1], Google App Engine [7] and Microsoft Azure [10] are among famous public clouds. There is also the possibility of using the Cloudlet or group of nearby mobile devices instead of a DS.

Several factors influence the quality of achieving the main goal of offloading,i.e. reducing the overall execution cost in mobile application. [69], [47], [112] analysed these factors and evaluated their effect on the offloading decision and its result. These factors are derived from the properties of mobile devices, application, environment and cloud as well as user dependent points(figure 2.2). Although the connection, mobile device and user behavior influence the offloading process, they

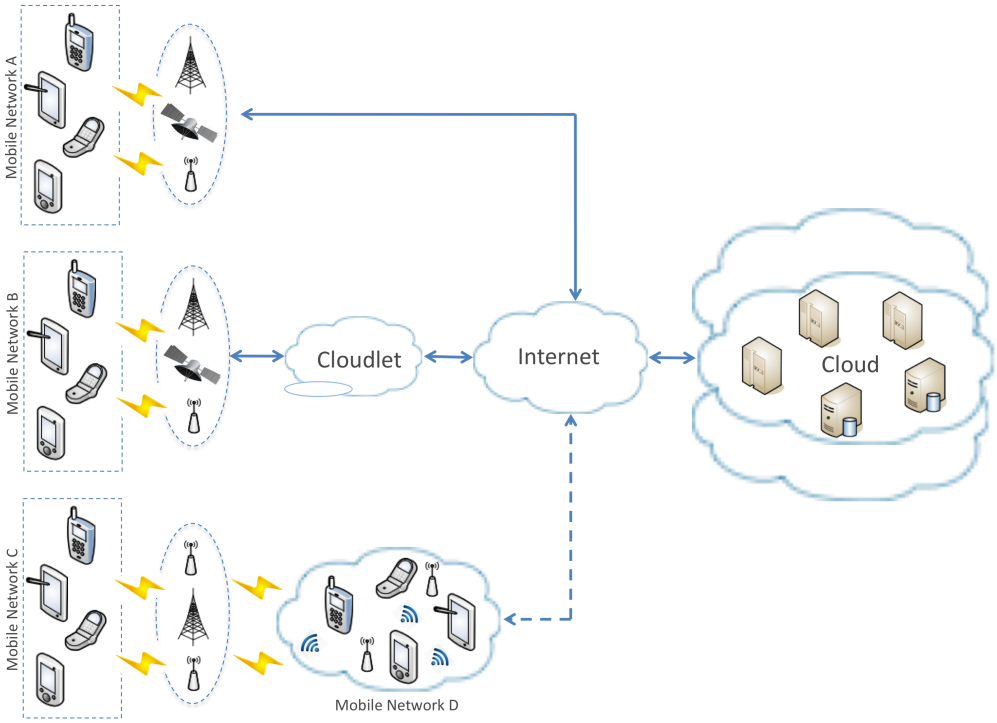

Figure 2.1: General architecture of Mobile Cloud Computing

usually cannot be changed by the programmer and are less emphasized in the various solutions offered in this regard. Application engineering and the way an application is made are among the issues that are taken into consideration. Granularity and partitioning in offloading are functionalities which have significant impact on the architecture, the manner of communicating with the server and the results of offloading. In the next section of this chapter, we will examine the relationship between the architecture and the relevancy of programs for cloud application engineering and then we will explain some examples of middlewares available with different architectures.

2.2 Architecture and Communication for Cloud Application Engineering

As mentioned in the previous section, MCC focuses on transfer of all or parts of a mobile application on machines that are more powerful than a mobile device and therefore, increases efficiency in mobile device. One factor affecting the quality of the transfer and its outcome is application engineering and the determination of what can be transferred and how. In fact, the process of offloading starts

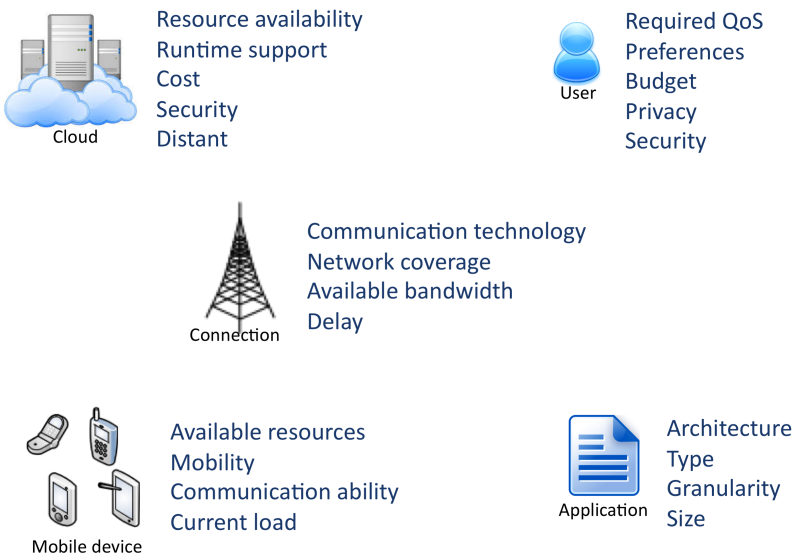

Figure 2.2: Affecting factors in Mobile Cloud Computing

with an application that is running on a mobile device and comes to an end in the surrogate. That's why an architecture based on application requirements is very effective in determining the transfer block. From the standpoint of applications engineering, the available architectures for mobile application can be divided into three models, Client-server based architecture, Virtualization based architecture and Mobile Agent based architecture. The following describes each of the three architectural models and introduces frameworks that rely on them.

2.2.1 Client-Server Based Architecture

In client-server based architecture, at first the application is divided into parts that can be offloaded. The division can be fine-grained or coarse-grained and is usually performed with the help of developer and by marking (annotating) the transferable parts. Then some of these parts will be transferred to the surrogate or server. In this method, protocols such as Remote Procedure Call (RPC) and Remote Method Invocation(RMI) are responsible for communication between the mobile device and surrogate. Stability of the protocols is one of the advantages of this communication

method. In addition, both of them well support APIs. RPC and RMI can be used only if these services are pre-installed and since such services may not be available on adjacent cloud or mobile device, the possibility of offloading may get limited. Spectra [49] and Chroma [21] are among the frameworks that have used RPC in offloading.

Spectra [49]

Spectra is a remote execution system that automatically estimates where and how the components should run. To that end, it monitors resource usage in small mobile device and resources available in the environment, especially static compute servers and takes into account the efficiency, energy consumption and the quality of the application. At first, the components that can be run on a remote server are statically specified and then the future resource requirements are foretasted and the manner of execution is proposed by resource monitor collection of Spectra at runtime with a continuous view of supply and demand of remote and local sources and by building models of resource consumption. Spectra forecasts the efficiency, energy consumption and the quality for every proposal and balances the conflicting aims during the selection process. Spectra has one of the earliest history based profiling cost models.

As for the Granularity, although fine-grained remote execution increases flexibility in spite of having access to other offloading options, coarse-grained is used in Spectra due to the possibility of increasing the efficiency with amortizing overhead over a larger unit of execution because overhead decisions cannot be overlooked. As a result, Spectra is suitable for applications that perform the coarse-grained operations, and therefore, not appropriate for applications with shorter operations.

For testing purpose, two main parts of the Spectra, the client and server, are run on a machine. Application makes remote procedure calls (RPCs) to local and remote Spectra servers. When Spectra is designing, the feasibility of using the service discovery protocol designed to identify available servers are dynamically considered, but since this feature is not supported, the potential servers are not statically stored in configuration file. When mobile device wants to offload a program, the Spectra client refers to the configuration file to obtain Spectra server specifications that are pre-installed with the application and act as a service. The Coda file system [72] is used to make synchronous the changes of files in different remote and local performances. Coda file system provides strong consistency when the network connection is appropriate and when the

network connection is weak. In fact, Spectra interacts with Coda to assure the remote operations would read the same data they read if they were run locally. Janus speech recognizer [124], Latex document preparation system and Pangloss-Lite language translator [51] are programs that have been modified to work with Spectra and evaluate the Spectra performance.

One of the advantages of Spectra is its good adaptability to the changes in the available resources of remote execution. Furthermore, the best execution option is recommended to the application. However, there are some points that can be improved in Spectra that include the lack of security features in data transmission, dependence on the programmer to determine the remotable parts of application, the lack of flexibility due to the fine-grain partitioning and overhead that are imposed to the system due to the use of Coda. As the number of servers increases, the amount of overhead increases too, but it is still an acceptable amount according to the results obtained with 5 servers.

Cuckoo [68]

Cuckoo is a complete computation offloading framework for Android with RMI like communication method and is based on partial application offloading in which the decision to remote or local execution of an application part is made at runtime. Cuckoo aims to simplify the task of developer for implementing the program. Cuckoo's integration with available development tools that are familiar to developers and automaticity of a large part of the implementation process have created a simple programming environment. Cuckoo is composed of a runtime system, a resource manager application and a programming model for developers which have all been integrated with Eclipse build system (figure 2.3).

The first stage of developing a program with Cuckoo is to develop a project and write the source code. The next stage is the separation of computation intensive (services) and interactive (activities) sectors with the help of existing activity/service model of Android via an interface definition language AIDL which is in turn done by the developer. Then, the Cuckoo framework generates an implementation of the same interface which includes a dummy method implementation at the beginning and should replace real methods implementations. Furthermore, the Cuckoo Service Rewriter CSR makes a stub/proxy for any AIDL interface. As a result, based on information provided by Cuckoo Resource Manager, the methods can be called in a local or remote form. In the end, the prepared code is compiled and the apk file is prepared and the user can install it on a

smartphone. The program can use any source on which Java Virtual machine is running, whether it is nearby infrastructure or cloud, as the destination of computation offloading. Of course, this remote resource should register its address in a part of the Cuckoo framework called Resource Manager that can be identified by smartphone. Cuckoo uses heuristics, context information and history to investigate whether it is effective to offload. The effectiveness can be adjusted to maximize performance or minimize the energy. To communicate with a remote server, the Ibis communication

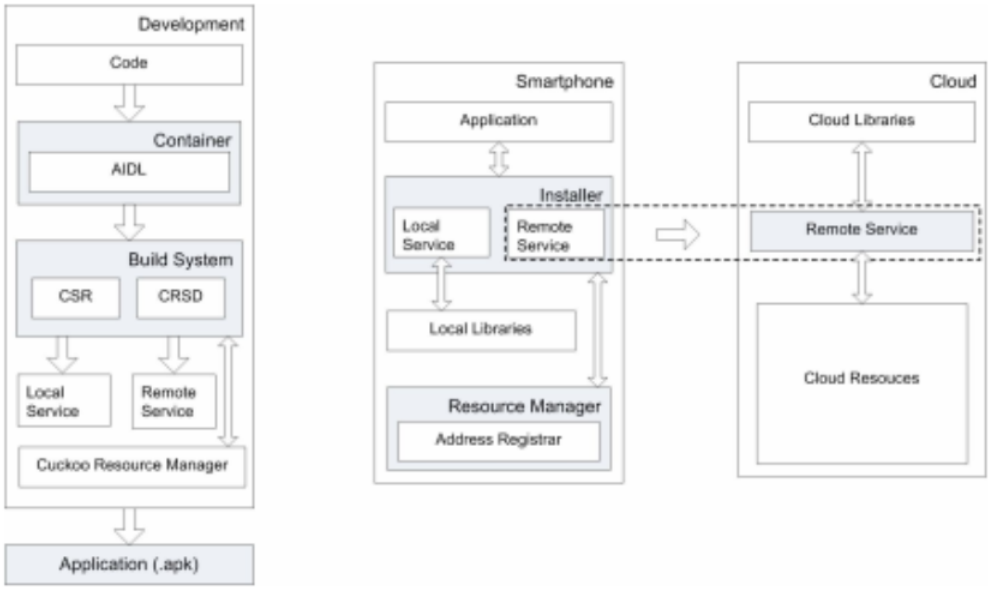

Figure 2.3: The architecture of Cuckoo [69]

middleware [122] which offers a service similar to RMI has been used. Cuckoo's performance has been evaluated with the help of two applications: eyeDef and PhotoShoot.

One of the benefits of Cuckoo is using the famous tools for application development and support of the partial offloading. Among its shortcomings, however, are a lack of Cuckoo's support from asynchronous callbacks and the state transferring from remote resources. Also, at switching time between remote and local execution, no situation is stored. In addition, the lack of security features for preventing the installation of malicious programs and controlling access to the server can be seen while designing Cuckoo. And finally, the decision of offloading in cuckoo is static and context unaware because the only thing considered in the context is the availability of the remote server.

2.2.2 Virtualization Based Architecture

In VM- based architecture, the programmer is not involved with offloading and has no idea of what is running on the mobile device and its surrogate and their management is undertaken by VM middleware. A virtual image is what is transferred in this method and is much larger grain than client-sever method, even in the case of coarse-grain. In a virtualization-based architecture, the image of VM is copied at runtime from the source to the destination. Live migration of virtual machines is a technique that is used to transfer the entire OS and its applications in mobile devices in some frameworks. The use of this method does not require a change in the program at the time of offloading. In addition, security needs increase due to the separated virtual spaces. But the transfer of large volumes of VM are generally regarded as challenging point of this method with regard with compatibility issues between overlays and bandwidth limitation. Most offloading frameworks use complete VM migration or a combination of it with partitioning algorithms. MAUI [32], MobiCloud [59], COMET [57] and Odessa [102] are examples of such a middleware. We explain below, the two most famous ones among them.

MAUI [32]

Between two general remote executing methods, application partitioning and full VM/process migration, MAUI [32] tried to benefit from both. It intends to decrease energy consumption in the smartphone using code offloading as well as to reduce programmer's interference.

Firstly, with the help of .NET programming environment features, two versions of the desired program will be made ready: the former is to be used for running the smartphone and the latter for being implemented on the infrastructure. MAUI makes offloading decisions at run time to do fine grain class/method level offloading. Firstly, offloadable methods are marked by the programmer in the programming environment provided by MAUI, and then remotable methods are determined automatically by the middleware using a combination of reflection programming and type safety.

At each method invocation, the optimization framework decides to offload the method if there is any server available. The cost of method offloading such as the number of states that must be transferred and the advantages of performing it such as the rate of decline in CPU cycle are among parameters that affect decision-making. Under server unavailability condition or for the definitive events, the method is run locally. Hence, the control of connection to the server and the

estimation of bandwidth and delay are performed continuously. All these parameters are considered as variables of optimization problems that are formulated and solved as integer linear programming to find the optimal solution. MAUI starts storing information after each method offload in order to make better decisions in the next performs. As shown in the figure 2.4, the main constituent

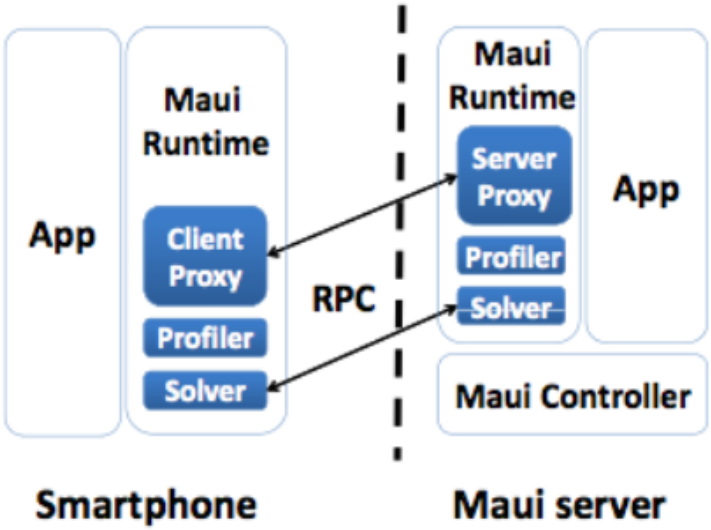

Figure 2.4: The architecture of MAUI

parts of MAUI are as follows:

- MAUI profiler: Offloading decision making depends on smartphone, network and program characteristics that are monitored and measured by MAUI profiler that includes three sections:
  - Device profiling: that is related to the measurement of energy consumption of the smartphone on the basis of CPU cycles. For this purpose, a hardware tool called a power meter is used.
  - Program profiling :the number of calls of each method as well as the number of CPU cycles required for implementation are the important parameters in the measurement of energy consumption which is calculated in this section.
  - Network profiling: used to profile the wireless networking environment to maximize energy savings. Therefore, a network measurement tool is applied to measure the round trip time and bandwidth.

- MAUI Solver: data collected by MAUI Profiler are used as input values of optimization problem in MAUI solver. A call graph is considered here with nodes indicating the methods, edges representing method calls and the consumed energy and runtime as the weight of edges. Then MAUI solver begins to solve the Integer linear Programming.

The results of tests on a face-recognition application, a highly-interactive video, a chess game and a real-time voice-based language showed that MAUI's energy savings and performance are impressive.

Odessa [102]

Odessa is a lightweight and adaptive runtime that aims to automatically and adaptively increase the performance and accuracy of mobile interactive perception applications by offloading, parallelism and pipelining. In fact, Odessa investigates the compliance of VM-based offloading and the level of parallelism simultaneously. Because of some of their requirements such as crisp response and continuous processing of high data rate sensors and also due to their being compute intensive, mobile interactive perception applications that conduct perception tasks, such as face recognition or object recognition by using the camera and other high-data rate sensors face some problems when they are run on mobile devices. Makespan and Throughput are two measures of responsiveness and accuracy of these programs. Offloading one or more compute-intensive application components on an Internet-connected server and parallelism on multi-core systems are two techniques that can be used to overcome this problem. Odessa aims to make optimal use of these techniques.

Tests show that changes in input variability, network bandwidth and device characteristics at runtime cause dramatic changes in responsiveness and accuracy; therefore, both offloading decisions and level of data or pipeline parallelism must be determined dynamically at runtime. Odessa is built on a distributed stream processing system called Sprout that facilitates the implementation and execution of parallel applications in addition to supporting the continuous, online processing of high rate streaming data. Sprout [99] features, i.e. the use of the data flow graph in programming model, automated data transfer and parallelism support are suitable to support Odessa runtime system. Data flow graph vertices and edges display processing step (called stages) and data dependencies between the stages, respectively.

To obtain low makespan and high throughput, fast response to input, device and network changes, as well as low computation and communication overhead are among the most important objectives of designing Odessa.

Odessa is composed of an application profiler and a decision engine. The profiler maintains the data related to the performance of applications such as execution time of each stage in the graph, wait time on every connector, volume of data transferred on each connector, and transfer time across the network connector edges. This section provides the data for decision engine without affecting application performance. Odessa estimates the bottleneck using these data and with the help of a greedy algorithm. Then the decision engine, based on a simple prediction and recent measurements of the network, examines whether offloading or increasing the level of parallelism in the bottleneck stage can be effective in increasing the efficiency. The decision engine functionality is divided into two threads on the mobile device, the former manages the data parallelism and stage offloading and the latter is responsible for the management of pipeline parallelism.

The costs for offloading and data parallelism are linearly estimated. Tests conducted on applications such as Face, Gesture, Object and Pose Recognition indicate the acceptable performance of Odessa. One of Odessa's positive points is the use of lightweight online profiler and simple execution time predictors, as well as the use of parallelism and pipelining, in addition to offloading. The other one is its rapid compliance with the scene complexity, compute resource availability, and network bandwidth. Although the security points are not considered and its function is limited to the perception applications.

2.2.3 Mobile Agent Based Architecture

A mobile agent is a software program with mobility, which can be sent out from a computer into a network and roam among the computer nodes in the network [27]. In an agent based model, the agents are not aware of the server, but know the locations towards which they themselves and other agents can move. The use of mobile agent compensates the lack of a standard APIs in the communication between different cloud infrastructures and heterogeneous mobile devices. In this way, codes and data are encapsulated within an agent in order to be transferred. Mobile agent places are virtual machines on which mobile agents run. The agents can also move between their places and communicate with each other. The management of these agents as well as the security

restrictions are among the weaknesses of this method.

Agent-based Optimization Framework [17]

In the offloading framework introduced in [17], offloading is done dynamically and at runtime with the help of autonomous agent-based application partitions. The goal of this dynamic performance optimization framework is an effective offloading that is done by enjoying the benefits of mobile agent computing, such as providing good support for mobile clients, facilitation of real-time interaction with server, ability of performing more robust queries/transactions and not required to preserve the process state.

In [17], mobile agents are developed using the Java Agent Development Environment(JADE) that supports multiple platforms such as Android OS. Each mobile application, statically and before it is installed on a mobile device, is classified into a set of agent-based partitions that can be offloaded on the cloud and components that must be run on mobile device due to their constraints. Agent-based application partitions are autonomous, i.e. they can move transparently between the cloud hosts without any need to be managed by the caller. Once the mobile application begins to run, the execution manager receives a list of machine instances in the the cloud from cloud directory service, and then having selected instances that establish the most powerful and fast communication with mobile device, offers an execution plan including offloading decisions. To make decisions, the execution manager uses a cost model and a static application profiler considering that if a partition is offloaded, all of its sub partitions should also be offloaded and partitions with by frequent communication should be kept together .

The results of the tests conducted on sudoko and NQueens Puzzle show that the proposed framework is promising for improved performance in terms of application execution time and energy consumption. Although this framework performs well and exploits the features of mobile agent, it depends on the developer to divide the program. A lack of attention to security tips and static profiling are among its weak points that can be improved.

2.3 Offloading Destination

Another factor that has a significant impact on the quality and outcome of offloading is its destination or its surrogate. As shown in the general architecture of MCC (figure 2.1), the destinations

of offloading can be divided into three different categories on the basis of the physical distance from the mobile device: DS that has the maximum distance from mobile device, cloudlet that is in the middle and a group of nearby mobile devices that are physically closest to mobile device. The processing power of a surrogate is inversely proportional to its distance from the mobile device, namely the closer is the surrogate to mobile device, the less it has processing power and vice versa. Therefore, the selection of the appropriate destination based on processing power and its distance from mobile device are of challenging points in MCC. The following section presents a brief explanation of how each of these three offloading types performs and introduces some of the frameworks in each category.

2.3.1 Distant Cloud Based Middleware

Sources used in this type of middleware are large collections of stationary servers that are located in the vendor or the company and that are recognized as public or private clouds. These resources that can be accessed via the Internet are highly available, elastic, scalable resources with high security features. While the efficiency and effectiveness of the methods that use remote cloud resources are strongly influenced by the long WAN latency caused by a long distance between the mobile client and cloud data center. In the following, two examples of offloading middlewares that use remote cloud for the implementation of remotable parts are discussed.

ThinkAir [73]

ThinkAir is an offloading framework that takes advantage of the smartphone virtualization in the cloud and provides method level computation offloading to support smartphone applications. Parallelizing method execution using multiple virtual machine images and focusing on the elasticity and scalability of cloud are of notable features of ThinkAir. Kosta et al. [73] tried to make a virtual image of the complete smartphone on the cloud and adopt online method level offloading to overcome the problems that exist in other offloading middleware such as the lack of scalability (in MAUI [32]) and the limitations of the program condition and of the environment (in Clone Cloud [28]). Building, resuming and destroying VMs take place in the cloud, dynamically and based on the needs. Consequently, ThinkAir can support parallelization.

As shown in the figure 2.5, the execution environment, application server and profilers are the

main components of the ThinkAir Framework which will be briefly explained in the following.

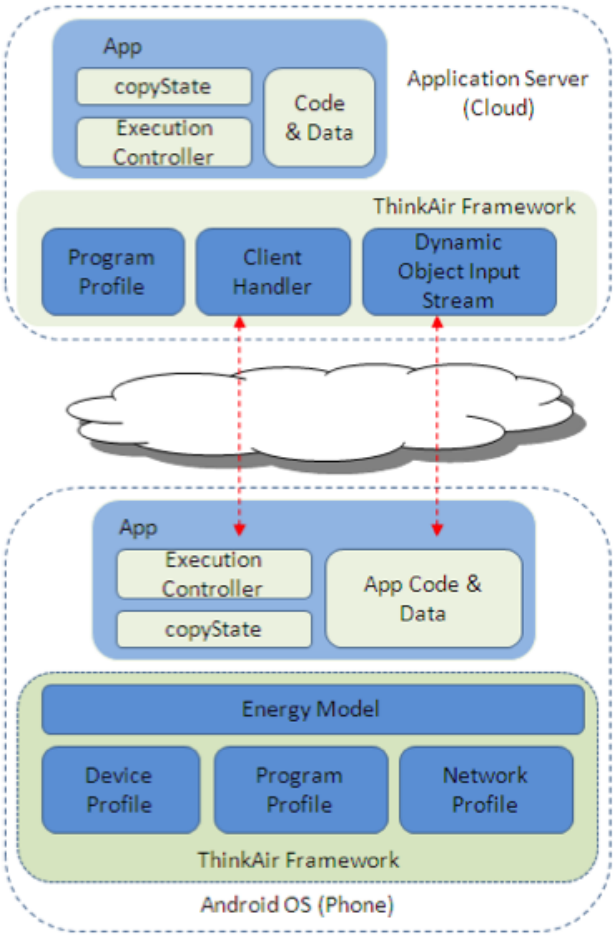

Figure 2.5: The architecture of ThinkAir

ThinkAir requires minor modifications to the code. Since the developer has indirect access to the execution environment, it can easily mark offloadable methods using the environment provided for him. Then, the ThinkAir code generator generates the new offloadable codes by using these marks. Management of transferable methods as well as offloading decisions to local or remote execution based on the current environment and the previous executions are undertaken by the Execution Controller. Execution time and energy are parameters based on which four decision policies are defined. The Application Server is responsible for managing the cloud of the offloaded code. Communication protocol execution, connection management, reception and execution of offloaded code and transfer of the results are all performed by the Client Handler. There are six VM models with different specifications for offloading that are managed by VM manager. If

necessary, more than one VM can be assigned to one task. The primary server is always online while secondary servers can be powered-off, paused or running. Parallel executions are supported by ThinkAir that is usually suitable for recursive algorithms and algorithms with heavy data volume. The profiling part is also of the utmost importance and its accuracy can lead to better decisions for offloading. Hardware profiler, software profiler and network profiler are three parts that deal with collecting information such as CPU usage, connection type, the number of calling methods, overall time of method execution and RTT in order to feed an Energy Estimation Model. ThinkAir estimates energy consumption and based on its estimation make offloding decisions.

Among the advantages of ThinkAir, it can be referred to taking into account of the energy consumption at the time of decision-making, supporting on demand resource allocation, and parallelizing implementation that reduces delays. In addition, ThinkAir compliance with the environment rapidly and effectively while environmental changes are taken into consideration at its design. In order to show these cases, programs such as image merging, virus scanner, face detection and N-Queens Puzzle are used for testing. Besides these positive aspects, changing the code with a modicum of programming that can make mistakes in marking methods, and the overhead that profile creates for a smartphone can be regarded as weaknesses of the framework.

Adaptive code offloading [50]

In order to benefit from the CC paradigm advantages including performance metrics, parallelization of tasks and elasticity in offloading which have been neglected in some offloading middlewares, [50] has introduced a fuzzy decision engine for thread level offloading on Android handset that considers dynamic variables of cloud in addition to mobile device variables. Furthermore, the decision making process has been strengthened with the help of evidence based learning methods based on a general understanding of mobile cloud infrastructure. This learning code offloading approach is able to turn raw code offloading traces into a knowledge that can be used to address the issues such as device diversity, adaptive application execution and unpredictable code profiling. This method uses fuzzy logic to determine when to offload. Therefore, with the use of variables derived from the mobile cloud architecture, a degree of accuracy is assigned to an offloading decision that can be analyzed on the basis of different intervals and rules. The information required to prepare and define the rules are provided by a mobile profiler and a cloud analyser that are updated pe-

riodically for local variables and asynchronously for external variables. Virtualization is a way to run offloaded components and code offloading traces are restored along with device information, application information and data component information. Figure 2.6 shows the fuzzy logic model used for code offloading that is, according to the usual form of fuzzy logic system (FLS), composed of four main sections of fuzzifier, rules, reasoning engine and deffuzzier.

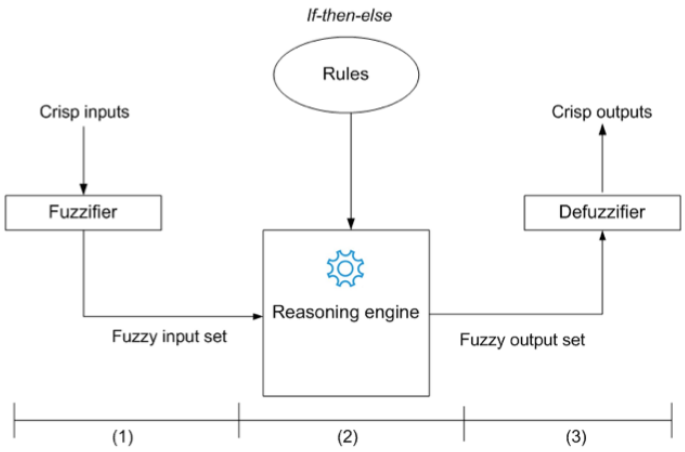

Figure 2.6: Fuzzy logic system

A crisp set is an input that is at first turned into a linguistic variable, and then analyzed in linguistic terms that are assigned to a specific membership function. The reasoning engine uses the input sets and builds an interface based on the rules and finally the output set is mapped on the crisp set. On this middleware, fuzzy sets include bandwidth, data transfer, CPU instance and video execution. For example, the network bandwidth can often be divided into intervals of low speed, normal speed and high speed distributed. *Remote processing = speed high AND data small* is a sample of the applied rules in [50]. It can be said that a Mobile Cloud Middleware framework covers the problems of interoperability across multiple clouds, transparent delegation and asynchronous execution of mobile tasks with the need to resource-intensive processing, a dynamic allocation of cloud infrastructure and Android mobile cloud messaging framework (decision engine is used in testing and it can be said that it is designed specifically for this task.) The delivery rate of Google Cloud Messaging which is the enhanced notification service provided by Google for sending asynchronous messages to Android devices is considered in the implementation of a video processing request to review the performance of this middleware and the results show that the grade of truth

is between 60 and 78 percent. The consideration of the strengths of the cloud in the process of offloading, decision offloading at runtime and the possibility of learning from previous performances are among the important points in this middleware. However with regard to the implementation of only some parts the proposed solution, it is not possible to be sure of the quality of its full implementation. Furthermore, the security of data transfer between mobile device and cloud is not considered.

2.3.2 Local Cloudlet Based Middleware

The offloading destination used in local Cloudlet-based middlewares is a collection of one or more resource-rich Ethernet-connected stationary computers that are usually located in public places. This group, though less powerful than the remote cloud, reduces the latency and network traffic because of the proximity to mobile device with a distance of generally one hop. "Cloudlet" is a name proposed by Satyanarayanan [110] for the proximate immobile clouds. In the middlewares such as [52] and MOCHA [42] Cloudlet is used.

VM-Based Cloudlets[110]

Cloudlet-based, resource-rich, mobile computing is the name given by Satyanarayanan et al. to the strategy used in [110] for the offloading process. In their proposed architecture, a resource-rich computer or cluster of computers called Cloudlet is used. It is available for nearby mobile devices and mobile device usually as a thin resource-intensive client, offload its tasks on this Cloudlet. Cloudlet physical proximity to the user in a 1-hop distance facilitates access to interactive response. Another advantage is the availability of the Cloudlet through low-latency high-bandwidth wireless connections. These features provide the opportunity of taking advantage of CC without the limitations of WAN, i.e.delay and long response time if Cloudlet is used instead of DS. Of course, when such Cloudlet may not be available near the mobile device, it can use a DS again to offload resource-intensive tasks. Virtual machine migration and VM synthesis are two methods applied in this article for computation offloading. However, the present article has focused on VM synthesis. During VM migration, the application execution is suspended, the state of the processor, disk and memory are stored and then the application execution is resumed exactly from the point where it has been stopped. Using VM synthesis, a small VM overlay that is derived from mobile device is

sent to Cloudlet. As shown in the figure 2.7, VM overlay is used in base vm to start the execution exactly from the point it has already been stopped. The feasibility of vm synthesis is demonstrated by the use of a prototype called prototype Kimberley. Among the advantages of Satyanarayanan's

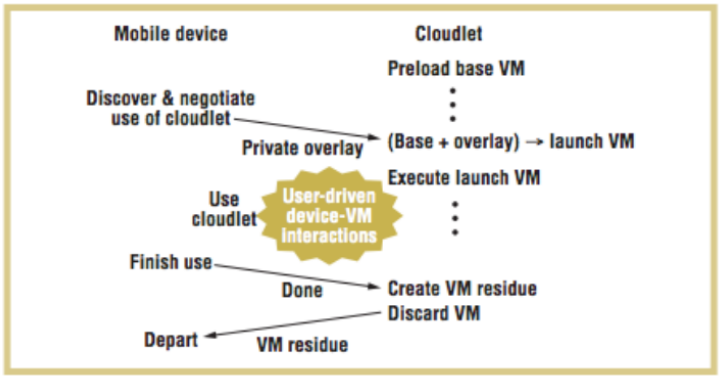

Figure 2.7: VM synthesis

model is the less fragility of vm-based model than alternatives such as process migration or software virtualization. Furthermore, VM-based models have less limitation and more generality than language-based virtualization approaches that require writing the programs in a specific language. Despite these strong points, when the user relies on the use of Cloudlet, the speed of VM synthesis becomes more important due to increased latency for service initiation. In addition, Cloudlet hand off must be equal to WiFi access point hand off, fast, invisible and seamless. Unfortunately, 60 and 90 seconds that is needed for VM synthesis is not enough for real-time tasks. Also, the power consumption and the amount of computation in mobile device increase when the overlay is extracted and compressed. The lack of a solution to increase security and protect the user from malicious VMs can also be seen. And finally, the proposed model is not scalable anywhere due to lack of Cloudlets.

From Mobile Devices to Clouds [52]

The possibility of code/task offloading in order to reduce workflows' energy costs is examined by [52] when mobile devices cooperate in a network that is equipped with Cloudlets. To define the problem, Gao et al. [52] have used two graphs. The first one is a directed acyclic graph that displays mobile workflow as a series of tasks and their relationships. The second graph displays a hardware platform on which the workflow runs in a way that vertices and edges denote processing nodes and

data links between them, respectively. Using two mapping functions, the mobile workflow graph is mapped on to a hardware graph. Once the objective functions have been modelled, a heuristic algorithm is used to create statistical and dynamic offload plans.

The algorithm proposed in [52] consists of two parts, the first part that implements on the smartphone helps to pick the best offload point, taking into account the environmental parameters in real time. The second part is located on the server. The key point of the decision making process is the trade-offs between time and energy. Every smartphone node is able to make decisions on itself based on the environment in which it is located. It is also considered that offloading two tasks on a Cloudlet reduces power consumption in the communication between the tasks.

The effect of the pairs of communication size/network connectivity and computation size/-Cloudlet processing speed on offloading decision is investigated using simulation. Different hardware and communication characteristics are attended in experiments. The results show that when the code repository is not available on the server, the large size of the executable parts has a negative impact on the ability to offload. The high volume of communication between the tasks also makes the offloading less feasible. The saving obtained from offloading is also directly related to hardware metrics.

2.3.3 Proximate Cloud Based Middleware

More recently, in some offloading approaches, mobile devices in the vicinity make their resources to run resource-intensive tasks available to each other. In this case that is based on the principles of CC, different mobile devices such as smartphones, tablets, notebooks or even IoT play the role of server for executing remotable parts of the application. The main advantage of the method is the physical proximity of resources to the mobile device, however the processing power is decreased compared with DS. Also, since it is more likely that mobile devices in the role of servers are damaged, lost or stolen, the method is less secure than the previous ones. Hyrax [83], virtual cloud provider [60], VMCC [67] and MOMCC [29] are among the middlewares that benefit their neighbours for offloading.

Hyrax [83]

Hyrax is a platform that is derived from Hadoop [9] which allows CC to be used on Android smartphones. Using Hyrax, client applications can easily apply a network of smatphones or heterogeneous phone and servers to perform their task. By changing the number of devices, Hyrax allows applications to make an abstract use of the distributed resources, regardless of the physical nature of the cloud. In fact, Hyrax makes it possible to use a cluster of smartphones as a resource provider and shows that this proximate cloud is practical and operational.

Apache Hadoop is an open source implementation of MapReduce [34] that provides a virtualized interface to a cluster of computers that have been randomly scaled. In Hyrax, a central server is responsible for coordinating the data and jobs of mobile devices and the relationship between smartphones is established via the isolated network of 802.11g. On the central server, just like a typical implementation of Hadoop, a NameNode and a JobTracker that have access to each client mobile devices are running. The central server only coordinates the data and jobs and does not perform any processing. Each mobile device instance of the DataNode and the TaskTracker runs the separate Android service. In addition, the threads which store the phones' multimedia data on the Hadoop Distributed File System(HDFS) and those which record sensor data run on the smartphone. JobTracker and NameNode are called by TaskTrackers and DataNodes and their response sent by Periodic heartbeat call and heartbeat response through RPC.

Sort, Random Writer, Pi Estimator, Grep, and Word Count which are derived from Hadoop examples as well as a sample application called HyraxTube are benchmarks used to evaluate the performance of Hyrax. HyraxTube is a simple distributed mobile multimedia search and sharing program that allows users to browse the videos and pictures stored on the network of smartphones and search them based on time, quality and location.

Among the strengths of Hyrax is that it can avoid the use of remote services to share data when data is available on the local network. It also has an acceptable performance in local peer-to-peer data sharing. Hadoop which is the base of Hyrax provides the features that are necessary for the MCC infrastructure. It also helps Hyrax, with its mechanisms, to support fault-tolerance. After all, due to high overhead imposed on the system as a result of running MapReduce, Hyrax is very heavy for the current smartphones. In addition, it is applicable only for the smartphones that are connected through TCP/IP sockets, while in real terms, all smartphones' IP addresses are not

without limitation or connected to a local network.

Virtual Cloud Provider [60]

Given to the fact that cloud resources are not always available or access to them is very expensive, Virtual Cloud Provider [60] suggests to build a virtual CC platform using nearby mobile devices. This framework is designed so that offloading occurs when both the mobile device that needs offloading and mobile devices in its vicinity are in the stable mode, i.e. they stay in the same area or follow the same movement pattern.

The offloading manager sends and receives jobs from one node to the other adjacent mobile device via the peer to peer connection and manages them, and in this regards, it enjoys the Application manager, Resource manager and Context manager's help. The first step of offloading is to make the changes necessary to prepare the application to be offloaded. For example, adding the capabilities of the proxy creation and RPC support which is done by the Application manager. Profiling and resource monitoring is the responsibility of the resource manager. The profile of the application is composed of the number of devices required for building the virtual cloud and the resources needed for offloading. The Context manager synchronizes contextual information and makes it available in some way for other processes. The context is the location and the number of near devices.

The framework's performance is assessed with the help of a prototype implemented in Java and based on Hadoop. Taking advantages of the pervasiveness of mobile devices is the main advantage of this framework but the inherent mobility is not taken into consideration. Given the dynamic environment and high mobility, the need for stable mobile devices that are responsible for offloading and mobile devices that produce the virtual cloud are of its limitations. The lack of attention to security issues and in particular authentication in the interaction between mobile devices is also one of its deficiencies.

2.4 Middleware Classification Based on Decision Making Process

In the previous sections, we classify application offloading middlewares based on the application engineering architecture and the communication model with the remote resource as well as their destination types. However, each of above mentioned categories has a significant impact on the per-

formance of offloading regarding to their characteristics, their choice is usually done statically and at middleware design time; where deciding for what and when to offload based on network connectivity, available bandwidth, available resources of mobile device and cloud, partitioning granularity, context or other affecting factors (figure 2.2) is done at runtime. Identification of the remotable parts that should be sent to the remote resource and its exact time is the most important part of every offloading middleware and its responsibility is to the decision engine. Different approaches apply several decision making algorithms to make offloading decisions in order to augment application performance in terms of energy consumption, execution time or resource consumption concerning different criteria. Figure 2.8 illustrates a thematic taxonomy of existing offloading middlewares.

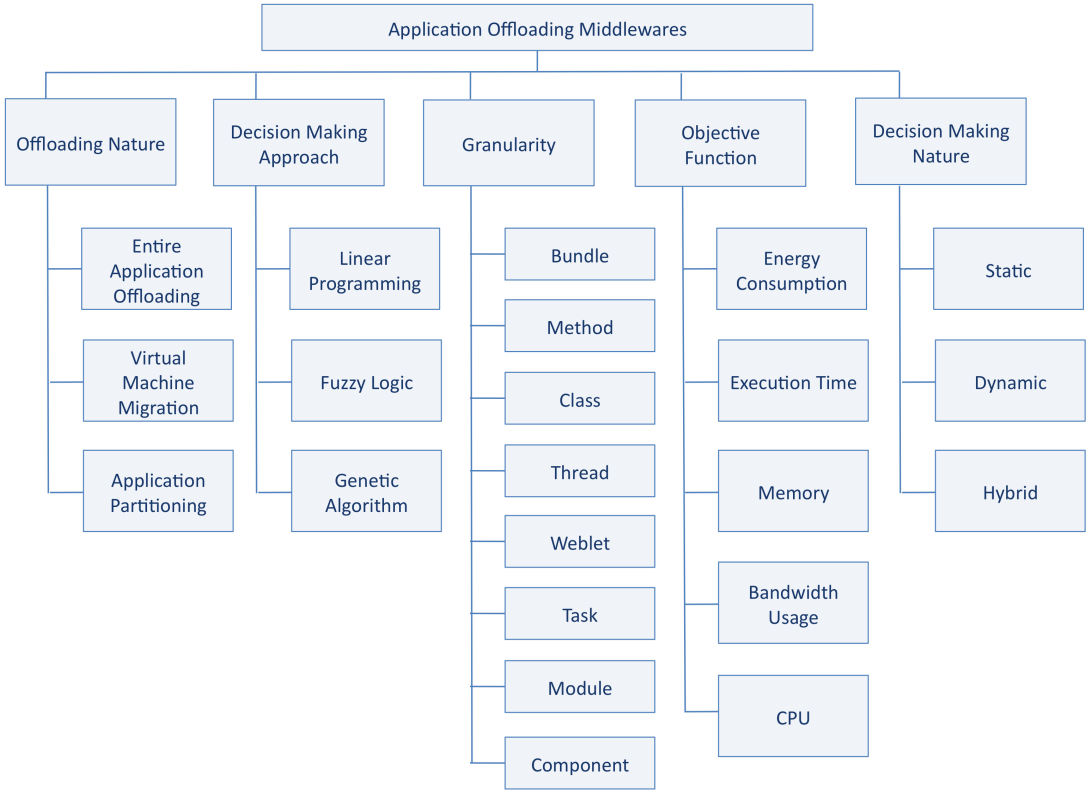

Figure 2.8: Thematic taxonomy of application offloading middlewares

The offloading nature indicates the main mechanisms employed for application offloading. Middlewares that apply VM migration encapsulate the running application into a VM instance of the mobile device and migrate it to cloud for execution. Entire application offloading means that the middleware offloads the entire processing job to cloud servers. Using application partitioning, the

Middleware	**Offloading nature**	**Decision making approach**	**Granularity**	**Objective function**	**Destination**
Spectra	Application Partioning	-	Component	Energy saving Reduce execution time	Cloud
Cuckoo	Application partitioning	-	bundle	Energy saving Reduce execution time	Cloud
MAUI	VM migration	ILP	method	Save energy Improve performance	Cloud
Think Air	VM migration	-	methods	Energy saving Reduce execution time	Cloud
VM-Based Cloudlet	VM Migration	VM Synthesis	-	Reduce end to end response time	Cloudlet
Hyrax	-	Distributed data processing	job	Reduce execution time fault tolerant	Mobile devices
Virtual Cloud provider	Partitioning	-	job	Execution time	Mobile devices
Clone Cloud	Application partitioning	ILP	thread	save energy seduce execution time	Cloud
Calling the cloud	Application partitioning	All/k-step algorithm	bundle	Minimize interaction latency	Cloud
Chroma	Application partioning	Tactic based	-	Reduce execution time	Cloud

Table 2.1: A classification of application offloading middlewares

resource intensive partitions of an application offload to cloud servers. Application partitioning could be performed statically either at compile time or runtime, or dynamically at runtime by following a dynamic evaluation of the current context and situation. These partitions that represent the offloadable parts of an application could have different granularity levels. For instance, a class level granularity indicate that classes of an application offload to cloud for execution. The primary objective of an application offloading framework is shown by its objective function. Saving energy and processing power are examples of the objective function. To meet these objectives, different middlewares proposed approaches such as linear programming, fuzzy logic and genetic algorithms. Offloading decisions could be made dynamically at run time, statically at development or a combination of these two. We compared some of the existing approaches based on the above mentioned taxonomy. The result of this comparison is illustrated in table 2.4

In this thesis, we propose a fine-grain application offloading middleware that applies bio-inspired

algorithms for making offloading decisions. The offloadable parts of the application could be executed either on DS or nearby mobile devices. The design and implementation of this middleware in explained in the following chapters.

Chapter 3

An Automated Application Offloading Middleware

3.1 Main Contributions of Designing Offloading Middleware 38
3.2 An Overview of Mobile Applications from an Application Engineering Perspective 40
3.2.1 Mobile Application Architecture 42
3.2.2 Mobile Application Transformation 44
3.3 An Overview of Application Offloading Middleware from a Runtime Perspective . 47
3.3.1 Design Objectives 47
3.3.2 Service-oriented Architecture for ACOMMA 50
3.3.2.1 Service Description 50
3.3.2.2 Service Interactions 52

In this chapter, we propose the mobile device to be a gateway to connect the IoT with the Cloud and a component of a Spontaneous Proximity Cloud at the same time. Like a DS, the Spontaneous Proximity Cloud could be used to defeat resource and processing power limitations of mobile devices via offloading. Our goal is to provide an application offloading middleware that responds to the challenging points defined in the previous chapter. We explain our main contributions for designing such a middleware and investigate mobile application and middleware architecture as the two main parts involved in offloading process. We first present our mobile application construction choices from an application engineering point of view as well as how to transform a normal mobile application to be ready to be offloaded by our designed middleware. Then, we introduce our middleware, explain its general service based architecture that makes it an easy to use open middleware and show how it respects our objectives: Making individual bi-

objective or collaborative decisions automatically to offload mobile application onto a DS or SPC while benefiting from a learning feature.

3.1 Main Contributions of Designing Offloading Middleware

Nowadays, we face incredibly small computing with embedded sensors in our everyday objects, that are close to user but suffer from a weak execution environment and greatly large with data and service clouds accessible anytime, anywhere but far from user. In the middle, there are mobile devices with the available resources and execution power neither weak as IoT nor powerful as cloud (figure 3.1). We consider mobile devices are set to become the universal interface between the IoT and CC worlds. Instead of their short battery life time, new generation of mobile devices are usually powerful enough for personal usage but may not be enough to be a gateway to close DS and IoT. We propose mobile devices to create a *Spontaneous Proximity Cloud* that could overcome resource limitations of IoT and/or nearby devices as their offloading surrogate.

A SPC is a collaborative group of moving mobile devices in proximity that its members occasionally join and leave. Geographically nearby mobile devices are in physical proximity while mobile devices with the same interest such as printer discovery are in semantic proximity.

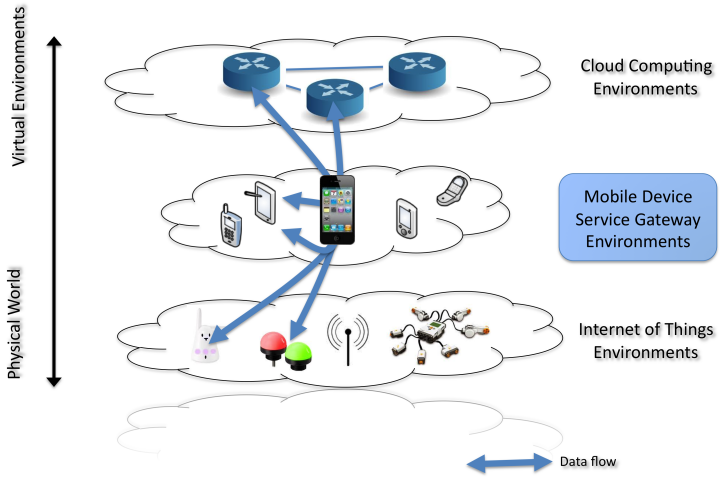

Figure 3.1: Mobile devices as IoT gateway and SPC

The main purpose of any offloading middleware is to use the capabilities of one or a group

of resource-rich machines for overcoming the processing limitations of a resource-poor handheld mobile device through delegating a mobile application totally or partially to execute on it. The mobile device can use remote resources in three different ways:

- Extending mobile device's access to cloud services: In this technique in which the cloud is often considered in SaaS (Software as a Service) format, computation and data handling are usually performed by the cloud. Software/applications as services provided by the cloud are accessed and used by users via the mobile device and often by using the web processors.

- Increasing processing power of the mobile device by total or partial execution of mobile application on the cloud: The cloud in this method is in the form of IaaS (Infrastructure as a Service) or PaaS (Platform as a Service) that increases the power and capacity of the mobile device by executing its resource-intensive or computation-intensive parts through code/application/computation offloading.

- Making mobile devices collaborate to provide cloud-like services: In this method, which is more appropriate for environments with ad-hoc networks with no access or limited access to the Internet or the cloud, a set of mobile devices in the vicinity constitutes a virtual mobile cloud in order to run their mobile applications with lower cost benefiting each other's facilities.

We are interested in the second and third approaches where the mobile device is responsible for its computation and data handling. We introduce ACOMMA, an Ant-inspired Collaborative Offloading Middleware for Mobile Applications, that makes adaptive offloading decisions at runtime using its bio-inspired algorithm. ACOMMA provides the possibility of classic offloading onto DS as well as collaborative offloading onto SPC. It also could make learning based offloading decision using already taken decisions either by the mobile device itself or its neighbour devices. Our main contributions while designing ACOMMA are as follows:

- Our first contribution is designing and developing an automated offloading midddleware that is easy to use for any mobile device without any special requirement by virtue of our proposed open architecture based on services. To respond to the issue of what to offload in a dynamically changing environment where the mobile device profile, context, and server properties play a considerable role in offloading effectiveness we propose a bi-objective decision making process

that easily re-adapts to environment modifications. We have also added a learning feature in the decision making process to avoid re-execution of decision making algorithms when there exists already taken offloading decisions in a similar situation.

- The second contribution is making collaborative offloading on a SPC. If many middlewares dealing with the issues of offloading, few proposed an approach in response to what and where to offload at the same time. In designing ACOMMA, we aim on offloading to the SPC and we improve our decision making algorithms to be able to decide on where to offload exactly between SPC nodes as well as what to offload in collaboration with other mobile devices.

The characteristics of a mobile application that needs offloading and the architecture of middleware itself have a significant role in achieving such a middleware and its performance and quality of offloading. In the next section, we explain how to model a mobile application firstly and then while mentioning different existing options for mobile applications from an application engineering point of view, we introduce and justify our selected options due to meet ACOMMA requirements. The section 3.3.2 is devoted to ACOMMA architecture.

3.2 An Overview of Mobile Applications from an Application Engineering Perspective

The main objective of ACOMMA, such as any other offloading middleware is benefiting from remote resource executing capabilities to overcome mobile device processing shortages. We are interested in partial application offloading; executing the mobile application in a distributed form between the mobile device and remote cloud resources. To this end, same as any other distributed system the application should be partitioned. Application partitioning is the task of breaking up the functionality of an application into distinct entities that can operate independently, usually in a distributed setting [119], [77].

Depending on the usage of application partitioning, there are different manners of performing it. For mobile application offloading in general, there are three models that could be used for application partitioning: graph based model, linear-programming based model and hybrid based model which is a combination of the previous ones, however there are some other approaches which do not fit in these three categorise. [78]

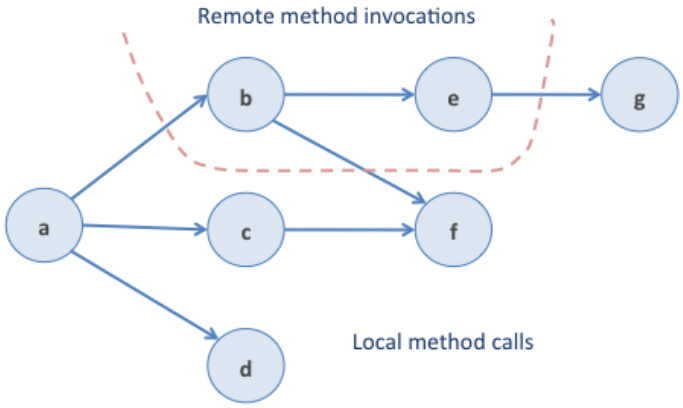

Figure 3.2: A sample call graph

We use a graph to model a mobile application where the nodes represent the application components and the edges show their relationships. Based on the interesting offloading granularity, these components could be tasks, methods, objects, classes etc... .

Flexibility and lightweight are two essential characteristics for a collaborative offloading middleware. To add more flexibility and lightweight to our middleware, we apply fine-grain application offloading at the method level so we consider mobile application modelled as a graph where vertices and edges represent methods and their dependencies in term of method calls respectively. Such a directed graph that shows a calling relationship between the procedures of a program is called a *call graph* in which loops imply recursive calls. A sample call graph is shown in figure 3.2. The graph partitions represent the executing environments of partition members. For instance in this graph there is just one cut that breaks apart two partitions where all methods execute locally except methods b and e that execute remotely on a distant execution environment.

Based on the main concept of application offloading in MCC the components of the application which are the nodes in the graph model and application methods in our method-level graph, could be executed either on the mobile device itself or a remote cloud. How an application is built from an application engineering point of view defines the way that its components on different executing machines should communicate, and following that the communication protocol and communication data format will be specified. In the rest of this section, we explain existing application architectures, communication protocols and data formats and their characteristics. Then we present our choices for mobile applications highlighting the advantages based on them we did our choices.

3.2.1 Mobile Application Architecture

The Choice of Application Architecture

Client-Server is the architecture style that we consider for designing mobile applications where a mobile device acts as a client and the cloud bears the server role. Between client-server model and virtual machine migration our main reason for choosing a client-server architecture is that it makes fine-grain application offloading possible. Although the fine-grain application partitioning is performed by some other approaches such as MAUI [32], offloading process based on virtual machine migration architecture influences fine-grain offloading because what is transmitted to the server is a virtual machine which is much bigger than mobile application partitions.

One of our contributions is making mobile application offloading possible between several collaborating mobile devices. These devices are limited in term of resources and there are also limitations at the network communication level. Fine granularity is important to make mobile device collaboration feasible because with shortages of resources and network bandwidth coarse grain application components may not be able to offload with good performance. It seems that more finer offloading granularity more flexible choices, and the more it is lightweight more higher the total performance. We are interested in fine granularity for both invocation and migration level so we choose the client-server application architecture that supports it.

In such a client-server model, an application component starts execution on the client side (mobile device), based on the offloading strategy the next component could be executed either on the mobile device itself or on the server if offloading is required. In this case, the remote component is invoked by sending a request with all required data. Execution will be suspended on the mobile device until receiving the remote component execution result. After that application execution continues normally on the client until the next offloading decision happens.

The choice of API

The request-response messaging pattern is used by the mobile device and cloud server to exchange messages. RMI (Remote Method Invocation), RPC (Remote Procedure Call), SOAP(Simple Object Access Protocol) and REST (Representational State Transfer) are options to choose as a communication protocol between the client and server. The most important factors for the choice of this protocol are its openness and simplicity of use. It makes the middleware usable for any

mobile device without any special requirement. This feature is necessary to have a collaborative offloading process between mobile devices in the vicinity. In addition, the openness of the applied communication protocol, simplify communication between the mobile device and sensor network when middleware works as a gateway between IoT and cloud. The communication protocol that is used in sensor networks is usually intra-network and can be used only by members of the same network. Since offloading middleware must be able to communicate with triple and heterogeneous environments of cloud, mobile network and IoT, it is necessary to have an open communication protocol.

Our choice for communication protocol is Representational State Transfer. Although REST, SOAP, RMI and RPC are interrelated options, they are not directly comparable. However, we try to explain the advantages of REST which are important and useful for our system considering its requirements.

RPC is to invoke a program procedure on a remote server. The main problem in RPC program is that the client will be tightly coupled with service implementation, which in turn causes problems when service implementation changes are required. The resource-oriented thinking of REST without getting involved in implementing how the relationship between the client and server is established makes REST simpler than RPC. Some researches investigated the advantages of REST over RPC. RMI is a java specific implementation of RPC that during its use we must ensure that class definitions remain in sync in all instances of the application and so even if only one of them changes all have to be redeployed. It also appears that when there is a firewall between the client and the server, the traffic between them using RMI will be blocked, while HTTP traffic and consequently REST are open in most firewalls. In addition, REST does not require a Java client and it can be regarded as one of its benefits when compared with RMI.

Compared to SOAP, REST is selected because it is more simple. SOAP relies on XML that imposed an overhead on the system, a feature that cannot be seen in REST. Of course, this overhead can be sometimes of benefits. It may happen that the SOAP being generic in using any transport can be regarded as one of its advantages, while REST uses HTTP/HTTPS. MCC, however, does not need to be generic because communication is established via the Internet and HTTP protocol. Furthermore, REST works very well when there are limitations in bandwidth and resources.

With the comparisons made, with due regard to the requirements of our offloading middleware

and also with respect to the researches that have considered REST to be more appropriate in mobile applications and mobile devices [58], we also use REST for the communication between the mobile device and the remote cloud server.

The Choice of Data Format

As a data format that should be sent over REST/HTTP from the mobile device to the remote server and vice versa, we chose two mechanisms of java serialization and JSON as the primary options. Java serialization is a mechanism that converts Java objects into a sequence of byte codes including specifications and data of an object that is suitable for transmission. Unlike JSON, this series is unreadable for human. JSON is a syntax for the storage and exchange of data whose prominent features are its language independency and self-describing feature. Although it seems that java serialization, due to its binary data, is faster in comparison with JSON.

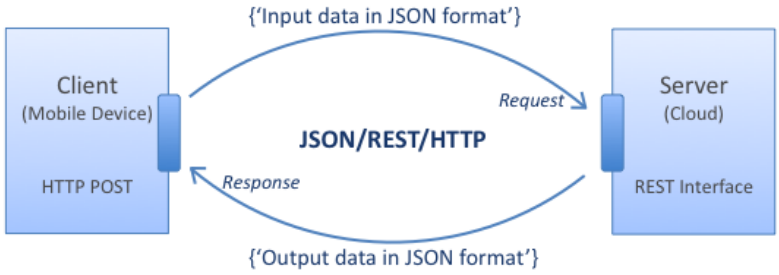

Figure 3.3: Mobile application architecture of ACOMMA from application engineering point of view

Figure 3.3 illustrates our choice of application architecture where a mobile device (client) sends its offloading request to DS (server) using the JSON data format and over REST/HTTP communication protocol.

3.2.2 Mobile Application Transformation

Since there may exist some components in any mobile application that have inherent dependencies to the mobile device and must be executed locally, an initial step to start an offloading process is defining offloadable parts of the application. Many offloading approaches rely on developer annotations to identify offloadable components. Its is clear that in this way the quality of offloading is

highly dependent on knowledge, expertises and experience of the developers who annotate applications. Any small issue in annotating may cause big changes in the offloading process. Some other approaches such as MAUI [32] and ThinkAir [73] have tried to minimize the developer intervention by applying rules for the detecting device dependent parts of an application. Our interest is to eliminate any need for manual annotation and modifications.

To coordinate with the client-server architecture and supporting REST/HTTP communication in service oriented model, there are also some modifications required in normal mobile applications. The application methods which are offloadable parts in our method-level application offloading should be modified in a form where methods act as services and are accessible via REST. In this thesis, *servicizing* is the process of changing methods into service form or creating *servicized methods*.

We designed a totally automated application transformer which distinguishes offloadable methods and then imposes servicizing process on those methods that are not limited to be run on the server. The process of creating a new mobile application with offloadable servicized methods that is adapted for offloading is a full automated process with no need of any developer or user intervention. The developer provides a mobile application normally without observing offloading tips. The transformer picks this application as input to create a modified application output. Furthermore, to be located on the server, a version of the application is provided that adds the accountability to services to any method. Depending on the circumstances, this transformer can be run directly on the mobile device or on any other machine and the modified application in output is transferred to the mobile device for executing.

Figure 3.4 illustrates the flow of a mobile application execution with transformer. The application transformer gets the source code and generates a new version of it with offloadable servicized method calls. This new source code is transformed to bytecode by the compiler and the virtual machine interprets the stream of bytecode as a sequence of instructions and then executes it to produce desired output.

In many cases, the source code is not available and only the binary version of an application is accessible to be installed on the mobile device. To handle such cases, we add an agent to the execution flow. This agent transforms the bytecode into a sevicized bytecode before the interpreting by virtual machine. We use JooFlux [65] for bytecode modification. JooFlux is a Java agent

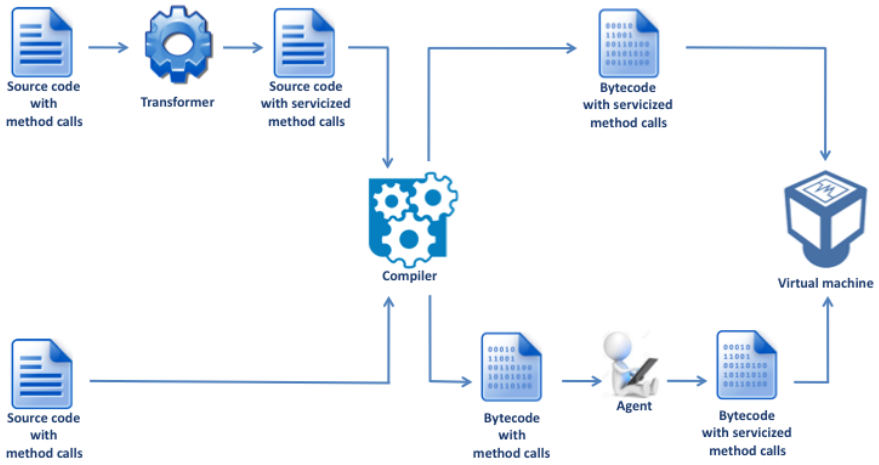

Figure 3.4: Mobile application execution flow with servicization modifications

for dynamic aspect-oriented middlewares that allows both the dynamic replacement of method implementations and the application of aspect advices.

The presented application transformer is an automated approach that dynamically converts method calls into services that can be used by the offloading middleware. This servicizing process can be done from both source code and bytecode which makes our transformer generic and flexible at input.

Although the ability to change the bytecode makes the approach more general and dynamic, it complicates the modification process and reduces the efficiency because the agent must be present on the mobile device. The availability of the source code, however, allows the application transformation process to be done on a system other than the mobile device and the bytecode of the modified application to be installed on the mobile device. Having the development chain before the mobile device increases the efficiency and performance, but decreases the dynamism while the system is no longer able to exert next changes during the execution.

The terms method and service are interchangeably used in the rest of the present study and we mean an application with offloadbale servicized methods by the words mobile application.

3.3 An Overview of Application Offloading Middleware from a Runtime Perspective

In the previous section, we modelled a mobile application with servicized method calls as a service graph. Now we focus on designing an offloading middleware to delegate this mobile application partially to execute on a DS or a virtual mobile network of nearby mobile devices to overcome the processing limitations of its resource-poor handheld mobile device.

We propose an open architecture based on services which is easy to use for any mobile device without any special requirement. In addition the inherent features of service oriented architecture provide some more benefits to our middleware.

- Location transparency is one of the main features of service oriented architecture that adds the ability of code mobility to it, i.e. the user can use the service regardless of its location.This feature is useful while offloading on different surrogates. In the midddleware collaboration mode, the mobile device can invoke a service from a DS or any mobile device of a virtual mobile cloud without any need to implement and execute a separate code for each surrogate type.

- The existence of a simple and standard format for accessing the services in service oriented architecture may make our middleware flexible and scalable. Thanks to this access format, each mobile device that is able to access the web services can make use of this offloading middleware without any need for specific changes and installation of new facilities. Any mobile device that supports web services could join the virtual mobile cloud to collaboratively execute its application benefiting from the pool of shared resources.

To design our service based middleware, we first review the important functionalities that it should have and then we propose the correspondent services to provide these features and functionalities. After that, we describe the proposed services and their interactions.

3.3.1 Design Objectives

As the main objective we aim to design ACOMMA, an offloading middleware that efficiently decides for:

1. which parts of the mobile application should remotely execute to improve performance in terms of two selected subjective?

2. where these offloadable components should execute? DS or virtual mobile network?

The effectiveness of the decision making process highly depends on how the middlewear deals with the issues of changing environment where the mobile device, communication network, context and cloud status may constantly change. We believe that to be a useful, practical and efficient offloading middleware which dynamically decides for above mentioned points concerning its dynamic environment, ACOMMA should be equipped with the following features and functionalities:

(a) ACOMMA should be aware of what the user needs

(b) ACOMMA should consider the application type in terms of resource-intensive, data intensive, computation intensive, etc

(c) ACOMMA should be able to find mobile devices in the vicinity and notify their hardware properties

(d) ACOMMA should know the abilities and features of a DS as well as its cost

(e) ACOMMA should pay attention to available networks and their coverage as well as communication conditions

(f) ACOMMA should be a lightweight and low consumption middleware

To this end we propose the decision making flow illustrated in figure 3.5. The availability of cloud resources to provision the required resources as well as mobile devices resources to execute the offloading process play a considerable role in offloading quality [20]. Similarly user dependent factors such as his preferences, limitations and requirements impact the decision making process. For instance, the application execution should be performed locally without benefiting remote execution or be terminated in the absence of enough local resources, if the user did not agree with offloading. The offloading process could also be terminated in sharp increase of latency, significant reduction of offloading quality or user security and privacy menace cases. The execution environment characteristics such as the distance from the mobile device to a cloud, network technologies and coverage, available bandwidth and etc, highly affect the usefulness of the offloading process [38].

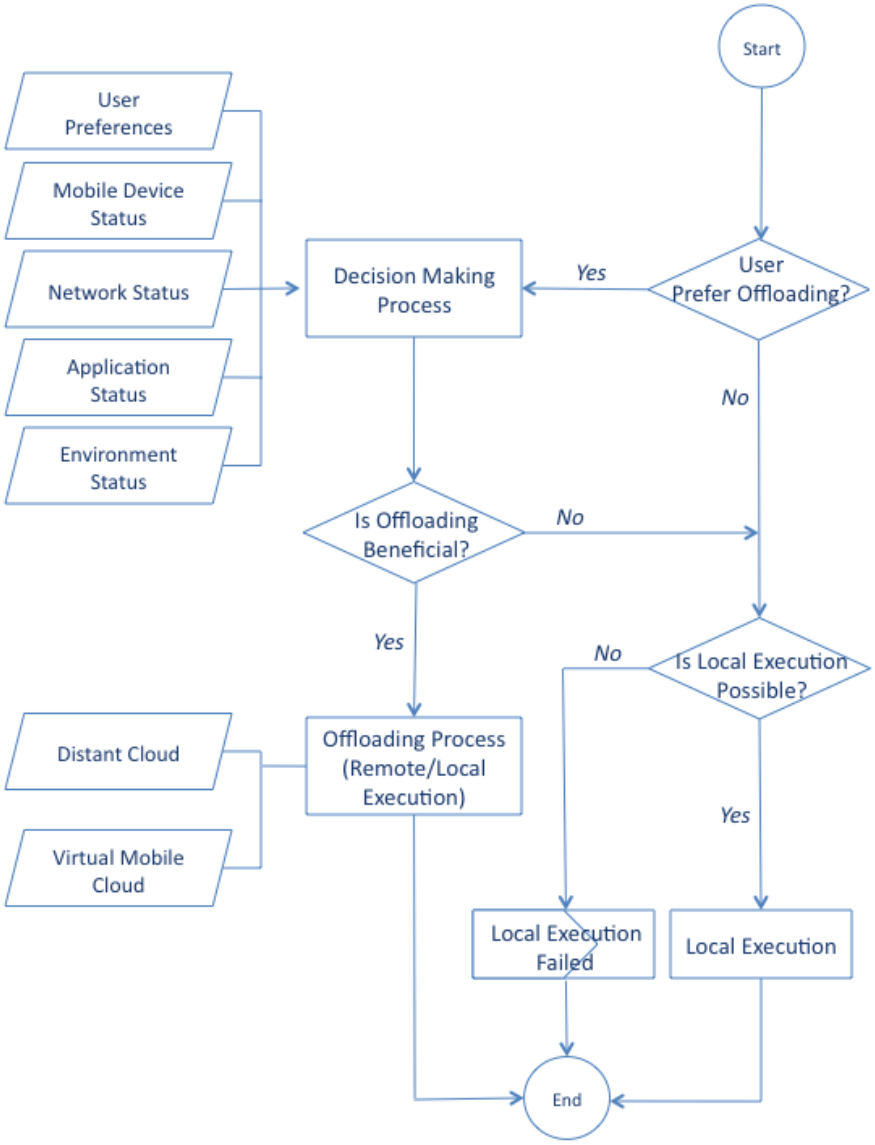

Figure 3.5: A general execution flow of offloading middleware

In addition, performing offloading on different applications do not result in the same performance. For instance, offloading a data intensive application in a low-bandwidth network affects performance due to the imposed large latency which is greatly different from offloading a computation intensive application in the same network status.

3.3.2 Service-oriented Architecture for ACOMMA

To fulfil the requirements and objective mentioned in the previous section, we propose a service based architecture illustrated in figure 3.6. In this architecture, all features are proposed in the form of services which could be accessed by any device that has the ability of accessing web services if needed. In the rest, we describe the various services and their interactions.

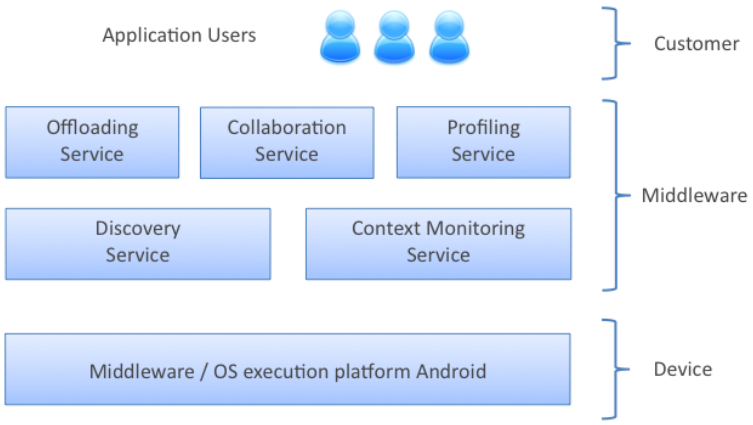

Figure 3.6: Layered architecture of application offloading middleware

3.3.2.1 Service Description

- **Offloading service**: This service that constitutes the core of our middleware, partitions mobile applications and determines offloadable parts that should be executed remotely. Considering the current condition of the mobile device, communication network and cloud, the decision engine decides for offloading focusing on performance improvement in terms of execution time, energy saving, etc.

  We are interested in bio-inspired decision making algorithm that could benefit from a learning feature. We develop an Ant Colony Optimization algorithm as well as string matching for decision making process. This process could be done individually by the mobile device itself or corporately in collaboration with nearby mobile devices.

- **Collaboration service**: This service makes the cooperation between several mobile devices possible. In order to overcome mobile device limitations, a group of mobile devices in the vicinity that prepare cloud-like services could be used as well as a DS. A mobile device can make use of the hardware resources of its adjacent devices by migrating its offloadable parts onto them instead of make offloading on a DS. This feature is more outstanding when a DS in not accessible or costly.

  Collaboration is also interesting from a data sharing point of view where nearby mobile devices share their made decisions with each other in order to enrich the learning database of the decision making process.

- **Profiling service**: This service creates a profile of user dependent points consisting of his preferences, limitations and requirements as well as application properties and usage form. Considering different places and conditions, the user may be interested in offloading or not, he may also not accept to use nearby resources for offloading. Various applications with different intensiveness affect offloading conditions. In addition, the same application may be used differently by different users. For instance, Facebook could be used for gaming, photo sharing, etc based on its users' interests and needs. A profile created by the profiling service consists of information about the user and application to perform offloading.

- **Discovery service**: This service is responsible for finding other mobile devices existing in the vicinity of the offloader device. Since mobility and dynamism are inherent characteristics of any mobile device, even if the offloader mobile device does not change its position for a while, in its highly changing environment there may be other devices that appear/disappear. Being aware of nearby mobile devices is an important point for a mobile device which needs to benefit from its neighbours. The discovery service periodically evaluates the mobile device's environment in order to find other devices in its vicinity. To have more accurate information about nearby mobile devices, there exit approaches that model the mobility trace of mobile devices and predict its next position.

- **Context monitoring service**: This service is responsible to give a context awareness to mobile device. Like profiling and discovery services, the context monitoring service oversights the mobile device's environment, although from another point of view. For this service, the

context consists of resource availability in mobile devices, cloud characteristics and communication conditions between the mobile device and an offloading surrogate. Information such as mobile device battery level, CPU usage and available memory, available resources on the cloud or virtual mobile network and their cost, available networks and their coverage, available bandwidth are prepared for the offloader mobile device using the context monitoring service.

3.3.2.2 Service Interactions

After giving a brief description of each component of the middleware in the previous section, in this section we explain how they interact between them.

- The offloading service is in relation with the context monitoring, profiling and collaboration services. A mobile application modelled as a service graph constitutes the main entrance to the offloading service. To make an efficient decision for offloading, this service requires information about the mobile device itself and its environment as well as user. Context monitoring and profiling services are responsible for meeting these requirements.

  Although the main output of this service is the partitioned call graph to determine remote and local execution, the results of this decision and the way the program is partitioned can be used in other devices to enrich its learning data for making collaborative decisions. Therefore the offloading service is closely related to the collaboration service. The history of partitions performed in different mobile devices of what is dislocated between offloading and collaboration service. After deciding for offloadable parts of an application, the offloading service communicates with either a DS or mobile network to execute offloadable parts remotely.

- The collaboration service has interaction with the offloading and discovery services. It could also communicate with collaboration services of nearby mobile devices. To this end, the collaboration service should firstly contact the discovery service to be informed of the identification of mobile devices in the vicinity. Then it will transfer the received data from other devices to local offloading service.

- The discovery service communicates with offloading and collaboration services. In addition to give neighbours identifications to the collaboration service, the discovery service feeds

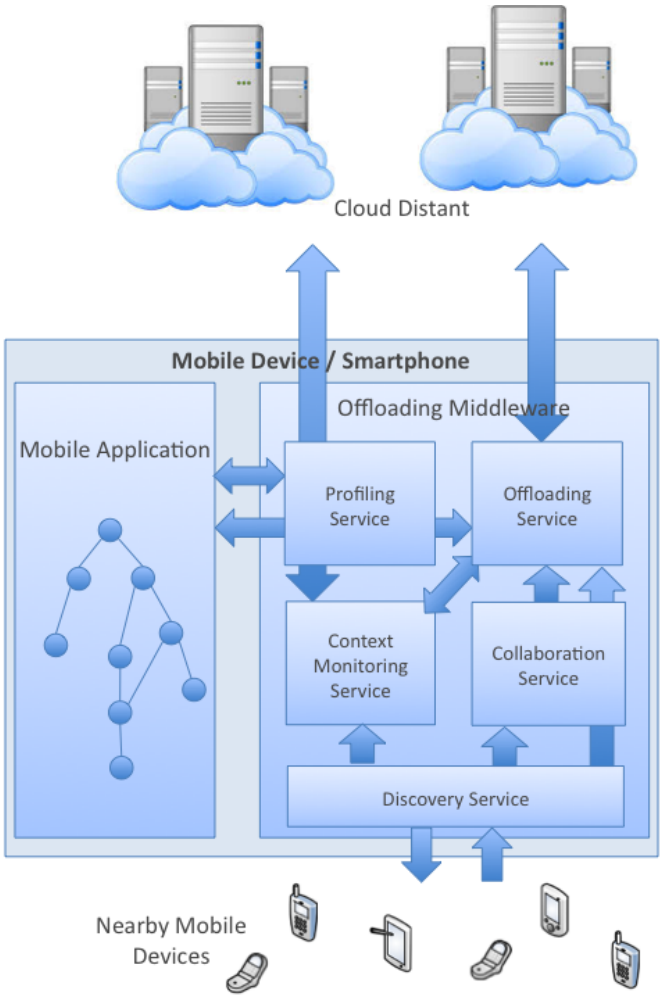

Figure 3.7: Service interactions of middleware architecture

offloading service with the same information. This information is used by the offloading service in a collaboration mode when it is deciding for the executing device of each offloadable part between mobile network members. The discovery service, similarly to collaboration service, is a communication bridge between the middleware and other devices from a communication point of view.

- The context monitoring service is in relation with the discovery and offloading services as well as cloud resources. It prepares the contextual information in terms of available resources of the cloud, the mobile device and other devices in the vicinity and the network status for the offloading service.

- the profiling service communicates with the offloading service to provide it user and application information. The profiling service is an internal service that does not have any relation with services on external devices.

After introducing ACOMMA and its architecture in this chapter, we focus on the offloading service and explain our proposed individual/collaborative learned based decision making process to make offloading depend on the context in the next chapter. Then, in chapter 5, we describe our proposed algorithms for the collaboration service as well as the discovery and profiling services where ACOMMA benefits from a SPC for both decision making and task migration.

Chapter 4

Individual Offloading Decision Making in ACOMMA

4.1 An Introduction of Decision Making Process for Application Offloading 55
4.1.1 Different Aspects of Offloading Decision Making 56
4.1.2 Application Partitioning Problem Considered as Shortest Path Problem . . 60
4.1.3 Solving Shortest Path Problem Using Bio-Inspired Algorithms 63
4.2 Decision Making Process of ACOMMA for Application Offloading 65
4.2.1 Application Offloading Flow of ACOMMA 65
4.2.2 Bi-Objective Offloading Decision Making Using Ant Colony Optimization Algorithm 67
4.2.3 Learning-Based Offloading Decision Making Using String Matching Algorithm 71

In this chapter, we describe offloading decision making as an application partitioning issue and explain how ACOMMA considers it as a Shortest Path Problem. Then we focus on our proposed bio-inspired algorithm to solve this problem. We justify Ant Colony Optimization as a suitable bio-inspired algorithm to determine where to execute application partitions. Finally, we introduce the learning based decision making process of ACOMMA benefiting from a String Matching algorithm.

4.1 An Introduction of Decision Making Process for Application Offloading

Is offloading possible and beneficial given the current situation? An affirmative answer to this question in fact allows us to discuss application offloading at first and the other issues after that.

To make a yes/no decision in this area, which is the most important issue in mobile application offloading, is the main responsibility of the decision engine. Hence decision making can be considered as the core section of any offloading middleware. The other blocks gain importance due to either the fulfilment of this section's primary needs or performing the next offloading step based on its decisions.

In the next stage, after the usefulness of offloading has been confirmed by the decision engine, questions such as *What to offload*, *When to offload* and *Where to offload* should be answered. From another point of view, although the usefulness of offloading is the key point that should be paid attention to, its answer depends on the answer to the second question set and every one of them may play a decisive role in the efficiency, quality and feasibility of offloading. Answers offered for the second set of questions form the basis of offloading middleware performance in conducting the application outsourcing process. It implies the significant role played by the decision making section in every offloading middleware. Although different middlewares, according to their intended approach, focus on finding answers to only somee f the questions, the decision making section constitutes the core section any way. In the following, how to answer these questions as well as the possible solutions are dealt with, and then our proposed offloading middleware and its used approach will be introduced from this perspective.

4.1.1 Different Aspects of Offloading Decision Making

What to offload

Among studies conducted on MCC and of different offloading middlewares that have been offered in this area, *what to offload* is the principal issue that has attracted most attentions. As mentioned in Chapter 1, VM migration, complete application offloading and partial application offloading are general answers obtained for this question but the decision in this case is made later by the middleware designer while designing the system architecture. What is relevant to the decision-making block is rather the time intended for the partial application offloading. Therefore the decision engine determines how to partition the application as well as which of these parts or partitions should be run locally on the mobile device or remotely on cloud resources. [78] classified application partitioning approaches into three general categories: graph-based, LP-based and hybrid.

- Graph-based application partitioning: In the graph-based approach, a directed graph is used for modelling the executive states, costs, dependencies, control flow and data flow in a way that vertices and edges of the graph show the parameters or context of an application. The number of vertices and edges of the graph may differ depending on the granularity based on which the application is modelled. In this case, the decision maker aims to divide the graph into two or several parts to be run on different resources. An appropriate graph can be useful in making appropriate decisions and affect the efficiency and quality of offloading. A variety of approaches of graph partitioning have been introduced and implemented in different researches. [125] for example, employed a parametric partitioning algorithm for application partitioning. Authors of [123] designed a multilevel graph partitioning algorithm.

  Graph-based partitioning approaches do not necessarily provide the best partitioning solutions and their efficiency depends completely on the application behavior such as being or not being modularized, but it decreases the coupling effect and migration cost in distributed application processing [78]. These approaches are not suitable for applications with numerous components because high resource overhead may lead to a decreased performance.

- LP-based application partitioning: A Linear Programming-based approach is a mathematical approach used for finding the best amount that can, considering the limitations, maximize or minimize an objective function based on power consumption or execution time. In this approach, the intended objective is at first formulated as a mathematical optimization problem and is then calculated using the technique of linear programming for the best options for achieving the objective. The obtained optimal solution forms the basis of application partitioning decisions as well as the offloading process.

  The LP-based approach is applied in a considerable number of offloading middlewares. Of course, there are various techniques for solving it. [130, 56], for example, made use of Integer Linear Programming (ILP) to solve optimization equations, while [115] and [129] used zero-one Linear Programming (0-1LP) and Mixed Integer Linear Programming (MILP) respectively.

  LP-based approaches are mainly characterized by the ability to find optimized solution for a given objective function, although solving such problems demands a lot of computational time [92]. Furthermore, the optimized performance in producing the most realistic parti-

tioning creates a lot of overhead because extra profiling and resource monitoring are needed. LP-based partitioning algorithms, however, are dynamic. Since they are lightweight in handling a large number of users, they could reduce the operating costs of the cloud [78].

- Hybrid application partitioning: This partitioning approach is in fact a combination of the first two approaches. Attempts have been made in this model to improve the quality of application partitioning through taking advantage of the strengths of graph-based and LP-based models. In MAUI [32], for instance, 0-1LP has been deployed to deal with graph optimization in the application call graph. CloneCloud [28], also, models the application as control flow graph before applying ILP optimization.

  Unlike the approaches classified in two former categories with almost the same performance, different hybrid-based approaches do not have any strength and weakness in common. For example, some algorithms endure less overhead for analytical techniques, while some others bear unnecessary overhead. It seems based on [78] conclusion, the most ideal hybrid algorithm is a combination of ILP and data flow graph.

There are, of course, some exceptions for application partitioning models that can be included in none of the above mentioned options. For instance, it can be referred to the approach proposed by [24] in which the concept of partitioning applications of J-Orchestra into units of dynamic updates is used.

We focus on what to offload in our offloading middleware as a key issue raised in offloading. We use service graph to model mobile applications and apply partitioning techniques to determine offloadable parts. Graph based application partitioning makes fine-grain application offloading possible that leads to flexibility and lightweight of offloading process. It also is more efficient compared with LP-based approaches because it consumes less resource [94]. Unlike many previous approaches that have made single-criteria offloading decisions, we propose a bi-constraint algorithm for application partitioning.

When to offload

As posed at the beginning of the chapter, the second question raised during application offloading is when to offload. Although it is of utmost importance to specify the time appropriate for a beneficial offloading, this issue is dealt with just by a few offloading middlewares. The changes in

network communication conditions such as bandwidth and latency, changes caused by the use of mobility, a sudden increase or decrease in CPU load in the mobile device and the variation of user inputs or how to execute the application that can influence its performance, are issues that have been neglected.

The general approach used for considering the mentioned items and finding the proper offloading time is the prediction of future conditions with regards to what happened before. Based on the predictions and with due consideration to the current situation, the decision engine can determine the best possible time for offloading or even disregard it. Among a few studies done on this issue are [54], [23], [127]. Since most of the changes result from the user motion, [71] presented a user motion model that allows the implicit prediction of the user's next state.

As for appropriate offloading time, our proposed middleware also takes decisions implicitly. In addition to its online, dynamic and context aware decision making process lead to takes into account the current state while offloading, the learning feature refers to using previous decisions to take new ones. Making bi-objective decision using CPU usage and execution time as objectives which include battery consumption and network conditions as well, leads to the involvement of internal and environmental conditions of the mobile device in the decision making process which forms the basis of middlewares' focus on when to offload.

Where to offload

The third important stage in the offloading process is to find the answer to the question where to offload and determine the appropriate surrogate - cloud/ cloudlet/ mobile network. In addition to the possibility of choosing between different clouds to offload and though the cloud itself consists of a collection of machines, it is possible in some cases to choose between cloud, cloudlet and mobile networks. Although a fixed server, cloud or cloudlet has been used as a surrogate in most studies on offloading, in few ones more options have been proposed to choose from while the offloading middleware is executing. [113], by the use of Dijsktra routing algorithm, presented a task allocation algorithm for the distribution of jobs on adjacent mobile devices. [45], also, exploited a collection of mobile devices as a cloud but dealt with them as a single surrogate.

We consider both a cloud server and the Spontaneous Proximity Cloud as offloading destinations and proposed a collaborative decision making algorithm to be able to choose appropriate destination

from several cloud servers, several adjacent mobile devices or a combination of the two considering the current situation.

Dealing with the issues of what, when and where to offload could be done statically at deployment, dynamically at execution time or a combination of the static and dynamic phases.

Although the static offloading decisions make it possible to use more complex heuristics, the risk of a wrong evaluation of the application behaviour and of the environmental conditions is augmented. [105] and [55] tackled the problem of static code offloading using offline profiling of applications.

On the other hand, although the dynamic performance can increase execution time and processing costs, it enhances the adaptability and accuracy, and consequently the quality of offloading. Dynamic approaches have attracted more attention recently [32], [28], [102].

Our proposed middleware, ACOMMA, dynamically deals with issues of what and where to offload as its main contribution while implicitly answers to when to offload. According to the studies conducted in this area, our middleware is the only middleware with such a broad functionality. In this chapter, we explain how ACOMMA deals with the issues of application partitioning to determine what to offload. Then in Chapter 5, we describe the collaborative decision making process to make offloading either on a DS or a Spontaneous Proximity Cloud.

4.1.2 Application Partitioning Problem Considered as Shortest Path Problem

In order to determine what to offload, the offloading decision making process of ACOMMA aims to perform graph based application partitioning as we mentioned in the previous section. We believe that the issues of graph partitioning and application offloading can be viewed as different classical problems:

- Classification problem: in which the application components are classified into two classes that are run on a cloud server and the mobile device or into several classes when several servers exist.
- Clustering problem: where the application components are required to be divided into two or several clusters to be run on server/ servers and the mobile device.

- Task assignment problem: each application component is regarded by this model to be a task which should be assigned to server/ servers and the mobile device.
- Multi-objective optimization problem: the aim in this case is to obtain the best partitioning based on several criteria. Therefore, a series of possible answers appears instead of an optimized one.
- Routing problem: it intends to discover the appropriate path for the consequent execution of application components by passing through the mobile device and server.

We propose the application partitioning problem investigate as a Shortest Path Problem where the application call graph is modified in a way that its nodes belong to at least two local and remote executing environments. The constituent nodes and vertices of the SP between the start and end points of the application represent the remote and local executions of each task. Since we have focused on a bi-criteria application partitioning, the problem is defined as a BSP problem.

The required modification of the service graph to transform the partitioning problem into a SP problem is illustrated in figure 4.1. In the transformation process, all graph nodes instead of start and end ones and consequently their coupling vertices duplicate. As the nodes represent the methods, the original nodes show methods on mobile device and the duplicated ones refer to the corresponding methods on the cloud.

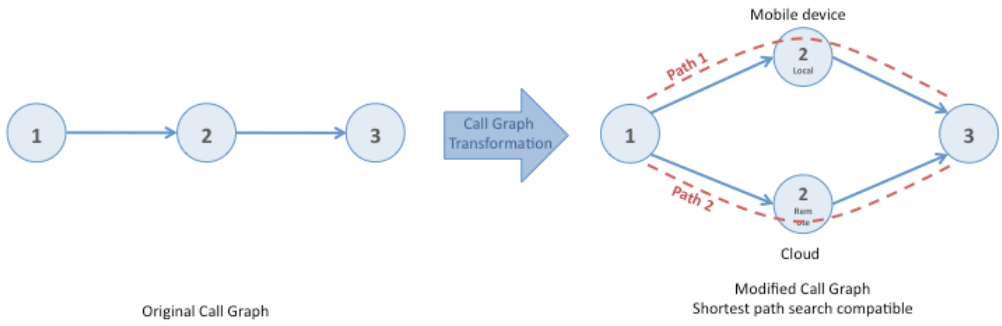

Figure 4.1: Transforming call graph to be compatible to SP problem

For instance in the original graph of figure 4.1, edges *1-2* and *2-3* show that *method1* calls *method2* and *method2* calls *method3* respectively. Assuming *method1* and *method3* to be start and end points that should run locally, the transformation process duplicates *node2* where the first one represents *method2* on mobile device and the second one is the same method on the cloud.

In modified graph *1-2local* and *1-2remote* edges represent local method call and remote method invocation of *method2*. The choice of the SP between *path1* and *path2* is in fact an offloading decision where they mean local and remote executions of *method2* respectively.

The goal of SP Problems is finding a path between two nodes in a weighted graph such that the sum of the weights of its constituent edges is minimized. As we want to take bi-criteria offloading decisions, there are two attributed weight for each edge. We consider CPU usage and execution time as constraints of the decision making and aim to find an offloading solution to minimize both of them. For the same mobile device and cloud, any change in the network conditions directly affects the execution time, for example, more network load leads to an increase in the execution time. There is also a direct relationship between the CPU usage and the battery consumption, the more an application uses CPU power, the more it consumes battery. Therefore, although it seems that the applied criteria are just the execution time and CPU usage, the network communication conditions and the amount of battery consumption also influence the offloading decision making process.

Figure 4.2 is an example of bi-weighted service graph where *Ti* and *Ci* represent the execution time and CPU consumption of *method i*.

In this graph, the choice of *start-L1-L2-R2-R3-end* as SP between the start and end points concerning the cost of each edge implies that in addition to the start and end points which are inherently local, *method1* should run locally and *method2* and *method3* are decided to execute remotely on DS.

ACOMMA decides for remote or local execution of each method at its beginning, whenever it is called and before its execution. In the graph, the edge between each local method and its corresponding remote method shows passing from local execution of method on the mobile device to its remote execution on a DS. For instance at the beginning of *method1*, ACOMMA decides where to execute it and the edge *L1-R1* is travelled only if *method1* should be offloaded. In this case, the local execution costs written in the input edge of *L1* are not involved in the total cost calculation.

Based on our knowledge this is the first time that an application partitioning problem in offloading decision making process is considered as a SP problem. Whenever we refer to a service/call graph in the rest of this document we consider a modified graph transformed in the above men-

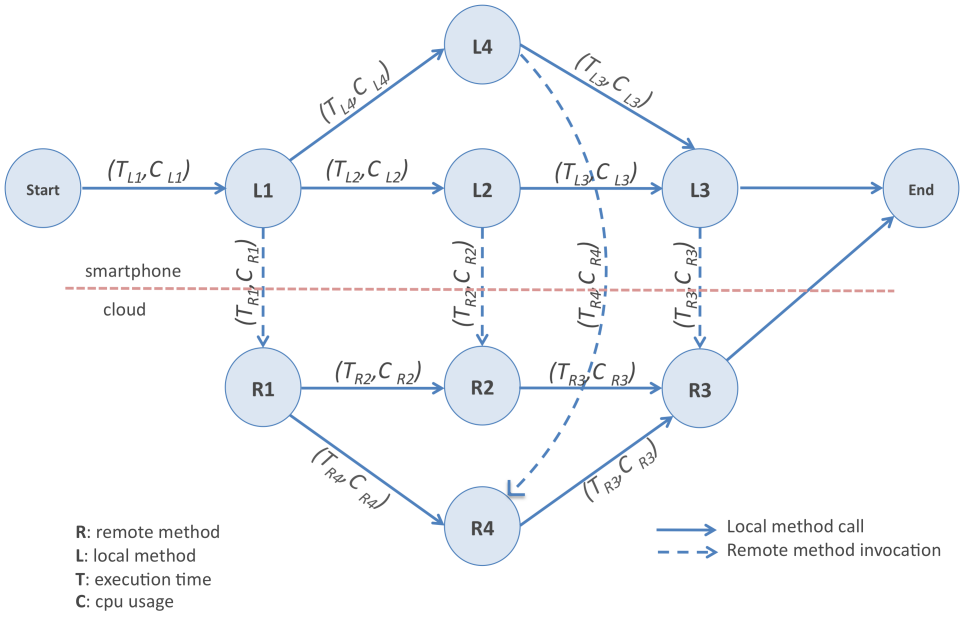

Figure 4.2: A weighted call graph with local and remote paths

tioned form. In the next section, we introduce our proposed algorithm to solve the SP problem in this graph.

4.1.3 Solving Shortest Path Problem Using Bio-Inspired Algorithms

Nature inspired computing (NIC) has emerged, taking inspiration from the nature, to develop new computer techniques for solving difficult problems. Biological inspired computing is also a subset of NIC that can be helpful in solving different problems. This approach has been widely used in CC and MCC. Having used a combination of genetic algorithm and fuzzy theory, [63] suggested an approach for job scheduling in CC. For the same problem, [81] made use of a mixture of ACO and artificial bee algorithm. [22] applied a modified ACO approach for service allocation and scheduling in MCC.

Recently, bio-inspired algorithms have attracted attention as an approach for application partitioning in a few number of offloading middlewares. [35] made some changes in a genetic algorithm-based optimization approach and used it for taking offloading decisions. In this approach, a pop-

ulation of solutions is used to find a globally optimized solution. Finding the optimized solution referred as an optimization problem, classically can be dealt with by exact methods (logical, mathematical, programming) and heuristic approaches. Bio-inspired algorithms are heuristic approaches composed of three general categories of evolutionary, swarm- based and ecology algorithms [25].

It seems that in complex problems, better results can be achieved using bio-inspired algorithms. Since many researches have published reports about the success of these approaches, it can be concluded that bio-inspired algorithms are among the strongest algorithms that solve the optimization problems [25]. In addition, the usage of bio-inspired solutions provides the opportunity to make competitive/collaborative decisions. It also improves capabilities such as self-organization and autonomy. The dynamic, robust and complex phenomenon with the capability of finding optimal solution is the trust behind bio-inspired computing [25]. As a result, we decide to use bio-inspired algorithms to take offloading decisions.

From different categories of bio-inspired approaches, we are interested in swarm-based algorithms. Swarm intelligence can be described as the collaborative conduct of a group of animals, especially insects such as ants, bees and termites, that are each following very basic rules but when seen in the field of computer science, swarm intelligence is a simulated way for problem solving using algorithms formed on the concept of self managed collective behaviour of social insects [38]. Here are some characteristics of swarm-based algorithms that make them well suited to achieve our middleware goals:

- Finding a solution with cooperative work between individuals in swarm-based algorithms could be useful for ACOMMA while making collaborative offloading decisions and/or executing offloadable application parts on a SPC.
- While designing and developing an automated middleware, we could benefit from decentralized and self organized coordination of individuals in these colony-based algorithms.
- In population based algorithms individuals, in their behaviour, take into account what their neighbour did. They move in the same direction as their neighbour while remaining close to them and avoiding collision [82]. In ACOMMA, this is useful to make learning based offloading decision.

Between different swarm-based approaches, we apply ACO algorithms to solve the SP Problem.

ACO is inspired by the behaviour of ants in finding paths from the colony to the food. It is a probabilistic method for solving computational problems, which can be reduced to finding good paths through graphs. We found SP finding in a graph is close enough to the behavior of ants when seeking their route to food. In addition, as far as our study shows, in spite of rapid progress in the field of bio-inspired optimisation approaches and specially swarm-based algorithms, although many approaches applied ACO to solve SP problems, no studies have claimed that one approach is superior to the others in solving the problem of the SP finding. Furthermore, the studies are not conducted in similar conditions and they cannot be relied on.

In next section, we explain the decision making block of ACOMMA and describe how it uses ACO as well as SM algorithms to make learning based offloading decisions.

4.2 Decision Making Process of ACOMMA for Application Offloading

4.2.1 Application Offloading Flow of ACOMMA

As an offloading middleware, ACOMMA is expected to deal efficiently with the issues of mobile devices' resource limitations benefiting from the power of a DS. It applies its decision making policies on a mobile application modelled as a service graph to determine offloadable parts of the application and then continue the offloading process by handling the remote execution of offloadable parts. We assume that a servicized version of s mobile application is installed on the mobile device before ACOMMA starts the offloading process. We also assume that the server side application exists on a DS. Figure 4.3 demonstrates the building blocks of ACOMMA involved in the offloading process as well as their interactions.

The offloading manager and decision engine are components of the offloading service of ACOMMA where context monitoring and profiling blocks correspond to the services with the same name. The decision engine determines the offloadable nodes of the service graph what their remote execution on the cloud results in performance improvement. The decision engine may simply make bi-constraint offloading decision concerning the current internal and environmental status of the mobile device prepared by the context monitoring service using an ACO algorithm. It may also make learning-based offloading decisions benefiting from its previous decisions in the same situa-

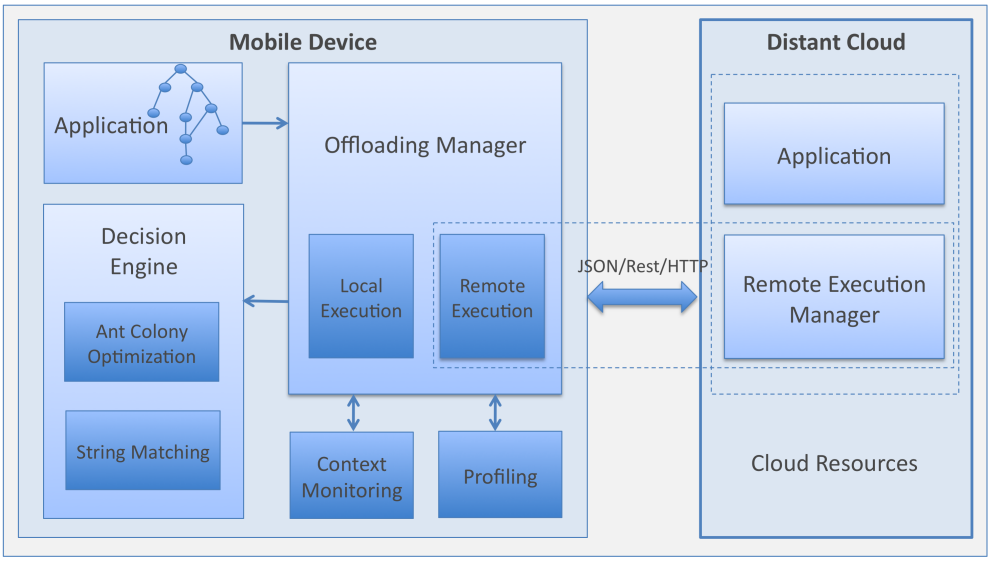

Figure 4.3: An architectural view of offloading building blocks in ACOMMA

tions archived by the profiling service with the help of a SM algorithm. At this step, there are just two available executing platforms for application execution: mobile device and DS.

The detail of the offloading process for each application method is shown in figure 4.4 as a flow diagram. During application execution, whenever a method call occurs, the offloading manager intervenes and asks the decision engine to decide about the remote or local execution of that method. Based on the requested decision making mode, simple or learning based, the decision engine applies ACO or SM algorithms to determine where to execute this method to increase the total application performance. Based on this decision, the offloading manager handles the method execution for local execution on the mobile device itself or remote execution on a DS.

If the decision is to run the method locally, the application execution continues on the mobile device; however, when a remote execution is required, the parameters needed for the method execution are converted into JSON data format by the offloading manager and transmitted to the server via REST/HTTP protocol. The execution of the application on the mobile device is suspended until the reception of the method execution output on the DS. The remote execution

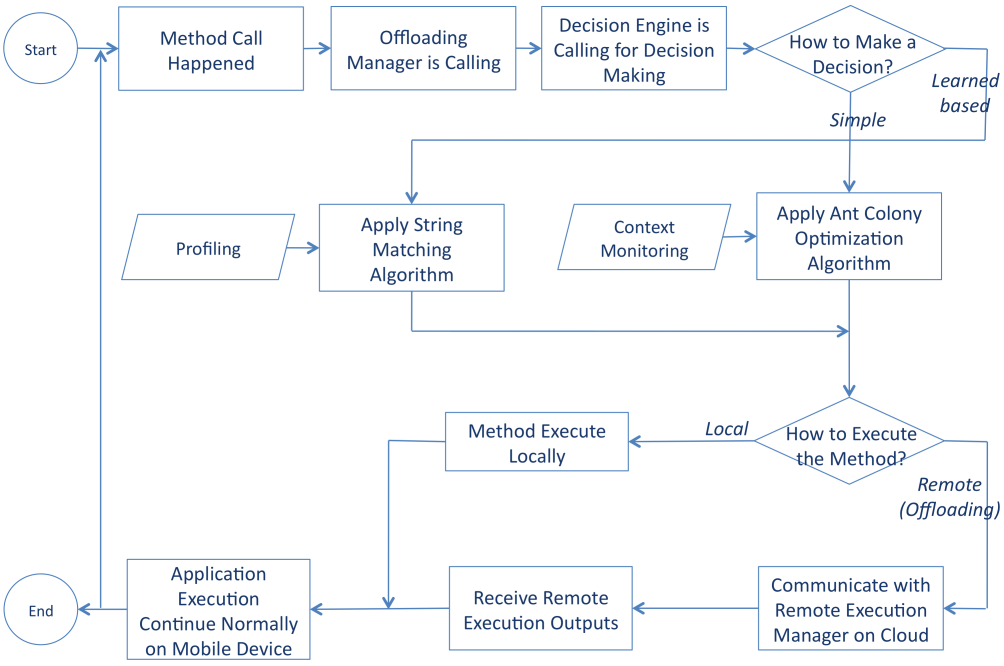

Figure 4.4: Offloading process in ACOMMA

manager in the cloud sends the obtained values of parameters to the corresponding method on the server side application and when the execution is over, sends back the execution results with the same format to the offloading manager on the mobile device. The local application execution resumes from a spot after method execution and after receiving its remote execution output. This is what happens for each method call during application execution.

After explaining the offloading process of ACOMMA, in the following sections we describe how ACO and SM algorithms operate respectively.

4.2.2 Bi-Objective Offloading Decision Making Using Ant Colony Optimization Algorithm

For the first time, [79] introduced an Ant Colony Optimization algorithm as a solution for the travelling salesman problem in 1991. To do so, they took inspiration of ants' behaviour when they search for the shortest possible path to get to their food source. In such problems that are often shown by graphs, an ant detects the best path to the food in a heuristic-based method using the

pheromone trail of previous ants. The pheromone augmentation after the same path has been travelled by the other ants and the evaporation of the pheromone indicate the learning desirability between the starting point and the food resource. In every stage, it is more likely that nodes with more amount of pheromone trail are selected. In recent years, procedures for updating the pheromone trail and evaporation as well as different transition rules have been applied in various ACO algorithms to solve problems such as the travelling salesman problem, telecommunication or vehicle routing problem, production scheduling, etc.

Due to the success of ACO algorithms in the single-objective areas [116], [41], made use of them, of course after a little modification, is taken into consideration to resolve multi-objective problems [62], [40].

Unlike single-objective optimization problems resulting in a scaler optimal solution, in multi-objective optimization problems the solution is a Pareto optimal solution set which is the set of all non-dominated solutions within the entire search space. Given the solution set of *x1* and *x2, x1* is a non-dominated solution if *x1* dominates *x2,* and *x1* dominates *x2* if *x1* is no worse than *x2* in all objective solutions and *x1* is strictly better than *x2* in at least one objective. In fact, multi-objective optimization algorithms focus on finding a trade off between problem constraints.

We are interested in bi-objective ACO to solve the graph partitioning problem with two constraints modelled as BSP problem. Coming to our BSP problem, a non-dominated set consists of the paths where the values of the objective functions are such that it is not possible to find another feasible path better than the current one in at least one objective function without worsening the value of at least another objective [31]. For example, in figure 4.5, there exist the four following paths between the start and end points, 'start-A-D-end', 'start-A-C-end', 'start-A-C-E-end' and 'start-B-E-end' with their respective related objective functions of (5, 5), (4, 5), (6, 6) and (6, 4). In this graph, theobjective functions (5,5) and (6,6) are dominated by (4,5), however between (4,5) and (6,4) the best path cannot be chosen because none is dominated by another. As a result, the non dominated paths are 'start-A-C-end' and 'start-B-E-end' with (4,5),(6,4) as objective functions.

The Bi-objective ACO proposed by [53] has been employed in the decision engine of ACOMMA to find the SP between the start and the end points of a mobile application modelled as a service graph. The pseudo-code of this algorithm is shown in algorithm 1.

Multi-objective ACO algorithms are classified into three general categories: the algorithms that

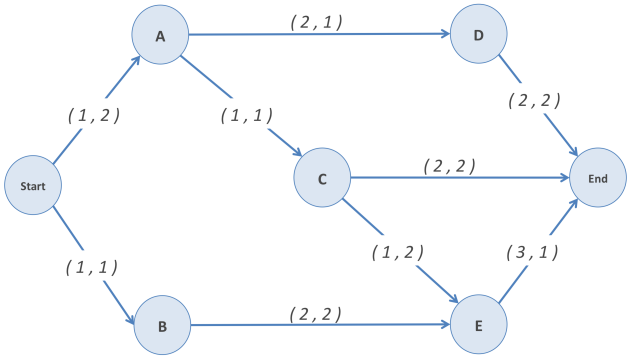

Figure 4.5: Non dominated solutions for Shortest Path Problem

use one colony for every objectives, algorithms that make use of one pheromone trail for every objectives and algorithms that apply a heuristic information for every objectives. The proposed algorithm in [53] belongs to the second category in which two pheromone matrices that are intended for two objectives are updated at the end of each iteration separately based on the generated results. In addition when an ant moves from one node to another, the pheromone trail is locally updated according to the evaporation rate. An artificial ant moves from one node to the next one based on a series of transition rules and with the help of two heuristic parameters.

We apply this algorithm on application service graph to have non-dominated set of SPs from start to end point of the application. Each path of non-dominated set may lead to different executing platform for each application method.

When an application starts, as an initialization phase, ACOMMA creates application's service graph in aforementioned form where both costs of cpu usage and execution time of each method is zero. For first method call, when ACO executes for the first time, non dominated set in consists of all possible paths. ACOMMA randomly select one of these solutions and execute application. Then weights of all edges of selected path update into real values of each method execution costs so in next algorithm execution, this selected path as well as the paths within common edges wont be in non-dominated set. After several execution all graph vertices will have real weighs progressively and ACO by applying local and global updates of pheromone trail gives different non-dominated

Algorithm 1 Bi-objective Ant Colony Optimisation

```
1: start
2: Initialisation:
3: init solution set, heuristic parameter and pheromone matrix
4: create new ant colony
5: loop:
6: an ant start from start node
7: ant move to next node using transition rule
8: make local update
9: if Ant reaches destination node then
10:     update non-dominated solution sets
11:     if All ants generate solution then
12:         make global update
13:     else
14:         goto 8
15: else
16:     goto 9
17: print non-dominated set
18: end
```

set at each execution.

ACOMMA runs ACO algorithm for each method call of application separately. However chosen non-dominated path shows local or remote execution of all application methods, ACOMMA applies this decision just for the current method and for the next one, it makes use of an ACO algorithm again to find new solution set. Making offloading decision for all the application methods when it starts may affect the context adaptability of ACOMMA as well as its performance while evaluating this path for the entire application at each method call makes the decision engine to take into account the mobile environment and results in dynamic decision making.

The nature of a method may cause considerable changes in the network or device state, for example due to the local execution of a method that consumes a large amount of resource, the available local resources on the mobile device greatly diminish and this may cause another decision for executing the next method differing from the previous one. In addition, considering a highly changing environment, there may happen great changes during a method execution, specially if its execution takes a long time and ACOMMA needs new a offloading decision for new conditions. For example, a high network traffic causes large execution time of the offloading method and new weights on the corresponding edge, so ACO execution may result in different solution paths.

We also assume that the executing thread continues on the same machine for nested methods.

So after starting the remote execution of a method on the cloud, its internal method could not be executed on the mobile device even if, based on the selected path, it should execute locally. So the offloading manager waits for receiving the execution result of the offloaded method including its nested functions and then restarts ACO for the next method call.

To avoid an execution of the ant colony algorithm for each decision making process and to make more accurate and efficient decisions, we propose using previous decisions for a current situation similar to a previous one. We explain this learning-based decision making process in the next section.

4.2.3 Learning-Based Offloading Decision Making Using String Matching Algorithm

Although offloading based on decisions made by an ACO algorithm leads to improving performance in terms of total execution time and CPU usage, the execution of the ACO itself imposes a processing load onto the mobile device. We aim to find a solution to decrease the ant colony algorithm execution cost as much as possible while profiling it for offloading decision making. To this end, we establish a learning-based decision making process that uses previous decisions made using the ACO algorithm for the same application and the same situation. We save the history of each application run as a string of executed methods and their execution platform (mobile device, cloud) and develop a simple SM algorithm to find the appropriate execution string in this history. We consider the decision saving as system training phase.

SM finds place where one or several strings (also called patterns) are found within a larger string or text [121]. Exact matching and approximate matching are two principal techniques in SM. We are interested in the former while developing ACOMMA.

The flow of the decision making process while using SM is illustrated in figure 4.6. Whenever an application starts, the offloading manager checks if the training phase has been done before. If not, the entire decision making process for this application runs inevitably using the ACO and the result save in cache. For application runs that happen after the training phase, for each method call, the decision engine searches whether there is any decision made at the same step of the application execution using the SM algorithm to execute the method in the same way as its previous execution. At any point, if the SM algorithm does not find any exact match, it means that there is no similar

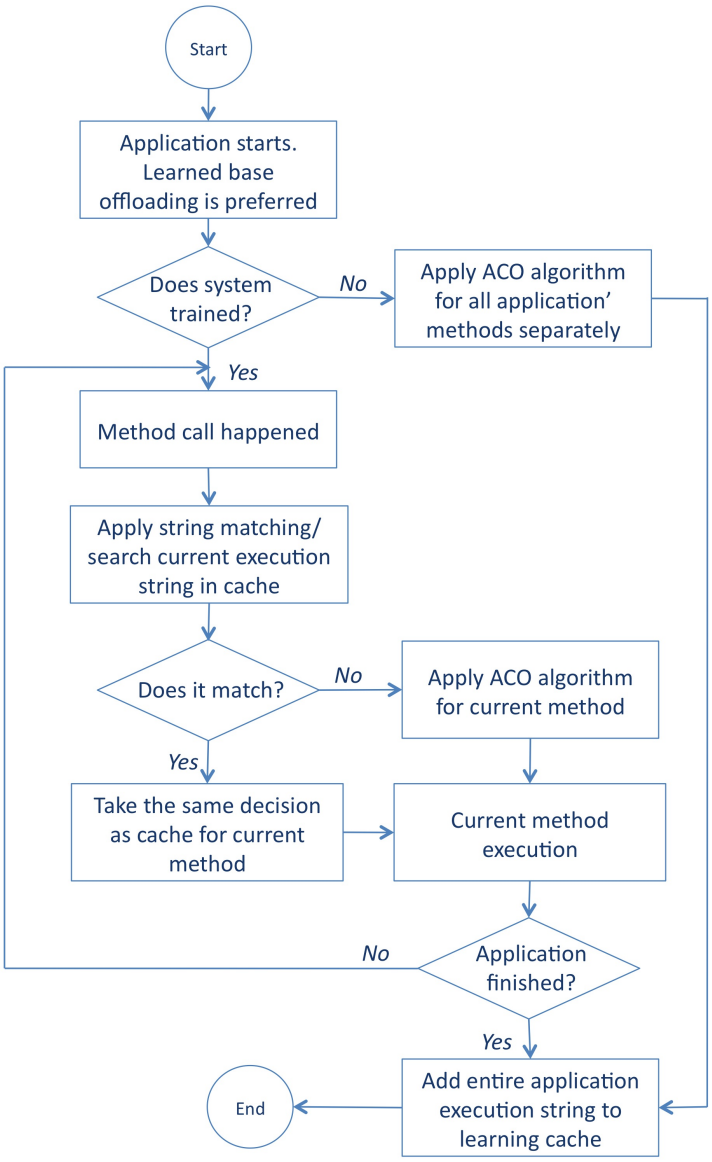

Figure 4.6: Decision making process using String Matching

situation saved in cache and the ACO algorithm takes the responsibility of decision making for the current method based on the current status. This sequence continues for all application methods and finishes by the end of the application run.

At any step, if there are several matches in the cache for a searched pattern, the decision engine may choose one of them using different policies. There are also some policies to apply for emptying the cache to avoid the exponential augmentation of its size that would result in more required storing space as well as more processing time for match finding.

Match search policies

Here are some simple policies for multiple matching cases.

- FIFO selection: Means that the cache is considered as a first in last out queue where the decision engine chooses the first matched string. To avoid one string to be always selected, it will be marked as used until all same matches are used once.
- Random selection: One on the matches option randomly select by decision making engine
- Weighted selection: The strings in the cache are weighted and the similar ones are selected based on their weight. The weight could be the total execution time of that run or the number of times that the application executes this way.

Cache invalidation policies

Some simple invalidation policies to avoid the cache to enlarge progressively are listed:

- Duplicate Prohibition: in this case, duplicate execution trails are not permitted to be added to the cache
- Periodic Invalidation: the cache gets empty periodically by applying this policy. This period could be a predefined time interval or a number of application runs.

We explain the procedure of SM with a simple example. Suppose that *a*, *b*, *c*, *d*, *e* are methods of an application, a_L means local execution of *method a* on the mobile device where a_R means its remote execution on the cloud. Figure 4.7 shows available cache. Again suppose that *method a* and *method b* already execute on local and remote respectively and the decision engine is in the process of decision making for *method c*. It searches for pattern a_L b_R in the cache using the FIFO selection policy.

There are three matches for this pattern in rows 2, 3 and 5. The first one is selected and *method c* executes locally, and row 2 is marked as used. While deciding for *method d*, the decision engine searches for pattern a_L b_R c_L and find two matches in rows 2 and 5 of the cache. Row 2 is marked as used so it selects row 5 and *method d* executes remotely. For *method e*, the decision engine selects row 5 again even though it is marked as used because it is the only match. At the end, the

1	a_L b_L c_L d_L e_L
2	a_L b_R c_L d_L e_L
3	a_L b_R c_R d_R e_R
4	a_L b_L c_R d_L e_L
5	a_L b_R c_L d_R e_L

Figure 4.7: A sample of String Matching Cache

string of this application run, a_L b_R c_L d_R e_L will be added to the cache with the current profile and execution time.

Although using SM algorithm could complete cache progressiveness, there is a little chance to find a new solution at each step. So we make use of a stochastic combination of ACO and SM algorithms for decision making. We can set in ACOMMA the percentage of each algorithm usage.

This learning-base decision making process could be used in a cooperative way while mobile devices collaborate to create a cache. The behavior of mobile devices' collaboration in ACOMMA is described in the next chapter.

Chapter 5

Collaborative Application Offloading

5.1 An Introduction of Collaboration-based Application Offloading 75
5.2 Collaborative Offloading in ACOMMA 79
5.2.1 Collaboration-Based Resource Sharing in Application Offloading 80
5.2.1.1 Creating Service Graph for Multi Destination Application Offloading 82
5.2.1.2 Applying ACO for Multi Destination Decision Making 84
5.2.2 Collaboration-Based Decision Sharing in Application Offloading 86
5.2.2.1 Collaborative Decision Sharing 86
5.2.2.2 Decision Cache Management 87

In this chapter, we start with the introduction and motivations of collaborative application offloading by using a SPC instead of a DS. Then we explain how ACOMMA provide collaborative offloading by illustrating its architecture. We consider that mobile devices collaborate either for using neighbours' resources as offloading surrogates or using their already made offloading decisions by cache sharing. We expand the usage of the ACO and SM algorithms for collaborative application offloading.

5.1 An Introduction of Collaboration-based Application Offloading

In the recent years and following the rapid development of MC and MCC, beside benefiting from clouds and cloudlets to overcome resource and processing constraints of mobile devices, making them collaborating to meet their requirements in term of resources attracted the attention of researchers.

Although the use of cloud resources can deal with mobile device's shortcomings in terms of processing power and memory, the physical distance between them could cause some problems and impose some expenses. The use of a cloudlet which is closer to the mobile device and its results demonstrate the advantages of using a close surrogate to mobile device while offloading. This is an incentive point for using even closer surrogate such as our proposed SPC or other cooperative mobile networks such as Transient Clouds [97], mClouds [85] and Mobile Device Cloud (MDC) [87], [88].

Although the use of mobile devices in vicinity as surrogates for mobile application offloading is still in its infancy, as an ideal for the future and though much research is still needed, there are significant factors that are encouraging for the entrance to such a category.

Cloud is not always helpful

Mobile devices differ greatly; the consumer market includes users with different requirements and budgets leading to great diversity in designing and manufacturing mobile devices in terms of hardware features such as processing power, available memory, and battery life time as well as sensors. Although DSs could fulfil mobile devices' needs for more processing power and exceeding memory by the help of application offloading, they might not be able to respond to all demands such as contextual information raised from local sensors. This information may be collected by the SPC. For instance, a mobile device that needs environmental information such as temperature and humidity could send a request to its nearby device that is equipped with the required sensors. In fact, the common context of mobile devices make their cooperation easier.

Cloud is not always accessible

- Regardless of the traffic in the network and the cost for accessing the cloud, its availability is also questionable. Despite numerous developments in the field of 3G, 4G communication networks, and the expansion of their coverage, access to the cloud is not possible always and everywhere. This lack of access, especially in certain areas and areas farther from the center of the city and also under special circumstances such as wars, earthquakes or other natural disasters that disrupted communications are more noticeable. Using peer-to-peer connections in the mobile network, for example through Bluetooth, without the need for backbone network can be useful when the cloud is not available.

- With the assumption of providing permanent access to the remote resource in terms of open communication platform, the communication with the cloud may not be possible due to its limitations. In addition to natural and man-made disasters that may damage the data center, significant technical failures such as that of the Amazon Elastic Computing Cloud (EC2) and Amazon (S3) cloud [15] can make cloud temporarily unavailable.

- Against the difficulties for accessing a cloud or even in the absence of a cloud or cloudlet, the high frequency of mobile devices and their increasing growth show that a group of mobile devices are always near each other in most cases. In addition to the increasing number of mobile devices, their average number for every user or household is also increasing [16], [70].

Figure 5.1 shows a motivating scenario where for instance nodes $C$and E may not reach neither cloud nor cloudlet resources. They are, however, able to collaborate/communicate with each other to run tasks that transcend an individual devices' capabilities. They may also be able to benefit from cloudlet/cloud resources by collaborating/communicating with node B.

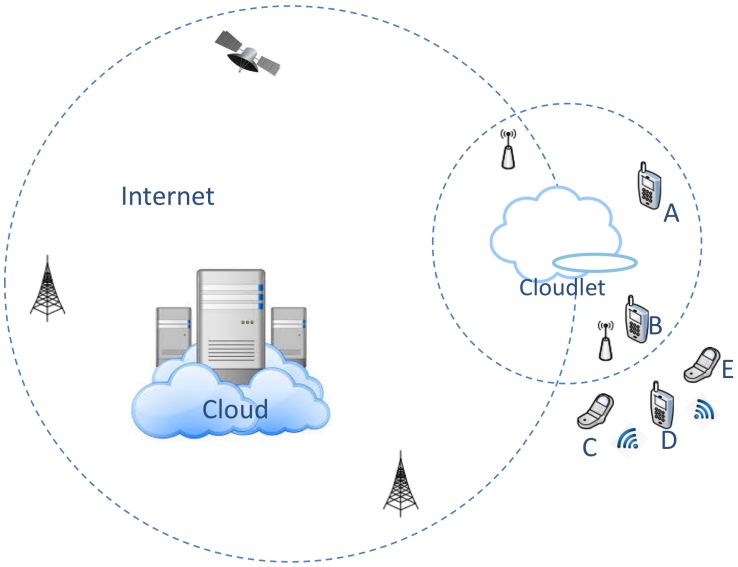

Figure 5.1: A motivating scenario to make mobile devices collaborate

Cloud is costly

- Economically, accessing a cloud imposes the cost of networking communication between the mobile device and the cloud besides the cost of the provider's resources. Although the net-

working cost varies according to the type and amount of user's usage and for different operators, it is important for the user in any way. Furthermore, although the cost of accessing cloud resources is not high, it is expected that the cost increases in the higher levels of uptime for better support, while the use of resource providers with a higher level of security and quality is more expensive [44].

- Another significant cost imposed by the use of the cloud and its communication with the mobile device is associated with the power consumption that absolutely contrasts with the most important goal of MCC that is offloading for saving power. Sending data to the cloud and receiving them not only demand a large part of bandwidth, but also lead to higher consumption of battery. Studies show that the energy consumption of a 3G cellular data interface (which is associated with the cloud) is 3 to 5 times much higher than WiFi transmissions (which can be used between mobile devices) [32], [85].

Cloud may be disadvantageous/ Offloading has side effects

- New wireless technologies, due to the continuous traffic growth, experience the shortage of capacities primarily. In this case, the use of a cloud for offloading necessitates a considerable part of the bandwidths to be allocated to the process of sending and receiving the data required for the remote execution. This considerable bandwidth allocation can lead to the production of overhead and increased efficiency of the network, while the use of a SPC can, in addition to the fulfilment of mobile device's needs, prevent the imposition of such an overhead which itself suffers from an additional load.

- Exponential growth of data centres and cloud infrastructures that lead to its ever-increasing energy usage are challenging points of the use of CC and therefore MCC that affect the environment in addition to creating economic challenges [66], [64]. In return, the use of pervasive computing and especially SPC is a perfect replacement for a greener computing where individual devices are powered by existing local renewable energy resources such as residential solar panels or wind turbines, or by harvesting the kinetic energy of the human body [15].

We find the above mentioned points strongly motivating for making use of a SPC for appli-

cation offloading. ACOMMA could take decisions to offload to several mobile devices benefiting from its adaptable decision making service. It creates a service graph concerning the number of proximate devices and applies its ACO algorithm to determine where to execute each part of a mobile application.

By the help of ACOMMA, in addition to using resources of a SPC, a mobile device could also enjoy other devices to make a mimetic collaborative offloading decisions. To this end, mobile devices share their decision cache instead of their resources and use offloading decisions already made by others for thr same application offloading and the same context.

In the rest of this chapter we explain how ACOMMA works with a SPC and supports collaboration of mobile devices for resource sharing as well as decision/cache sharing.

5.2 Collaborative Offloading in ACOMMA

To be able to benefit from a SPC either for resource sharing or cache sharing, a mobile device needs to communicate with its neighbours. In chapter 3, we proposed a service based communication to make ACOMMA usable for any mobile device that supports services without any special requirement. To communicate with a SPC, the mobile device uses the same communication protocol as with the DS. It sends and receives data in JSON format with the REST protocol over HTTP. There is just a conceptual difference between communications to DS and SPC (Figure 5.2).

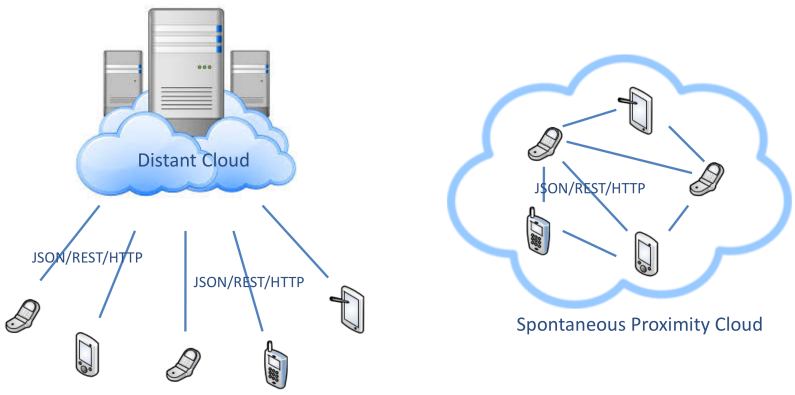

Figure 5.2: Centralized client-server vs. decentralized peer-to peer communication

A DS is a centralized system and mobile devices use a client-server model to communicate with it while a SPC is a peer to peer network in which two or more mobile devices (peers) pool their

resources and communicate with each other in a decentralized system. In peer to peer networks, the clients provide and at the same time consume resources. In such circumstances, there is no need for high availability in mobile devices because in case of unavailability of one of the SPC members, the resources shared with other members can still be exploited.

To make such a communication and collaborate with others, a mobile device needs to know devices in its vicinity as the first step. We developed a discovery service that helps ACOMMA to find the mobile device's neighbours. Service discovery works in a decentralized way. The mobile device broadcasts a message about its proposed service as soon as it joins a SPC and in response it receives the port numbers as well as information on the proposed services of all devices in the vicinity.

The building blocks of ACOMMA for supporting collaborative offloading and the communication between two mobile devices in the vicinity that are using ACOMMA as well as their communications with DS is illustrated in figure 5.2. The collaboration service of a mobile device is in charge of making nearby devices to collaborate using the neighbours information prepared by the discovery service while the offloading manager is responsible for offloading to a DS. The context and profile information could help ACOMMA to choose the SPC if the DS is not accessible or affordable.

The execution flow of ACOMMA in collaboration mode is shown in figure 5.4. After discovering its neighbours, and based on the selected mode for resource or cache sharing, ACOMMA applies different algorithms to make use of its adjacent devices. In the following, we firstly focus on benefiting from a SPC instead of a DS for offloading some parts of the application. We give a short history of researches already done in this domain and continue with how ACOMMA deals with this issue. Then we concentrate on the decision sharing process while using the decision cache of other mobile devices in the vicinity.

5.2.1 Collaboration-Based Resource Sharing in Application Offloading

Among lots of proposed offloading middlewares, only a few of them have focused on the use of adjacent mobile devices as offloading surrogates. Transient clouds [97] employ the collective capabilities of nearby devices that form an ad-hoc network to meet the needs of the mobile device. A modified Hungarian method has been applied as an assignment algorithm to assign tasks to devices that are

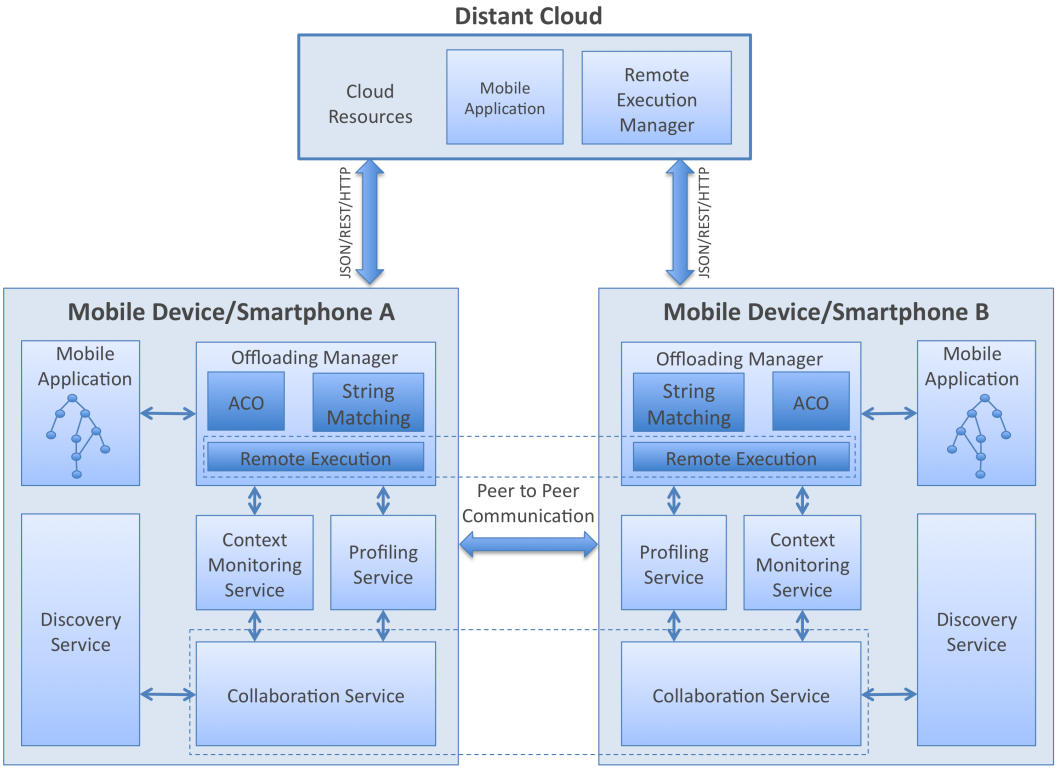

Figure 5.3: Building blocks of ACOMMA for collaborative offloading

to be run according to their abilities. The execution of each task by any device imposes some cost and the assignment algorithm aims to find the minimum total cost assignment. In this regard, [97] has proposed the dynamic cost adjustment to balance the tasks based on costs between devices. Miluzzo et al. [85] have suggested an architecture named mCloud that runs resource-intensive applications on collections of cooperating mobile devices and discuss its advantages. Kassahun et al. [15], however, have gone a step further and have formulated a decision algorithm for global adaptive offloading. They have implemented the program components on mobile devices set to optimize Time to Failure (TTF) while taking into account the limitations of the effectiveness of the program. Having highlighted the benefits of collaboration for mobile task offloading, Mtibaa et al. also implemented computational offloading schemes to maximize the longevity of mobile devices [87], [88].

ACOMMA allows a mobile device to offload onto adjacent devices. To this end, after discovering nearby devices, ACOMMA creates a modified version of the service graph for which the application partitioning problem for offloading decision making could be modelled as a SP problem. Then

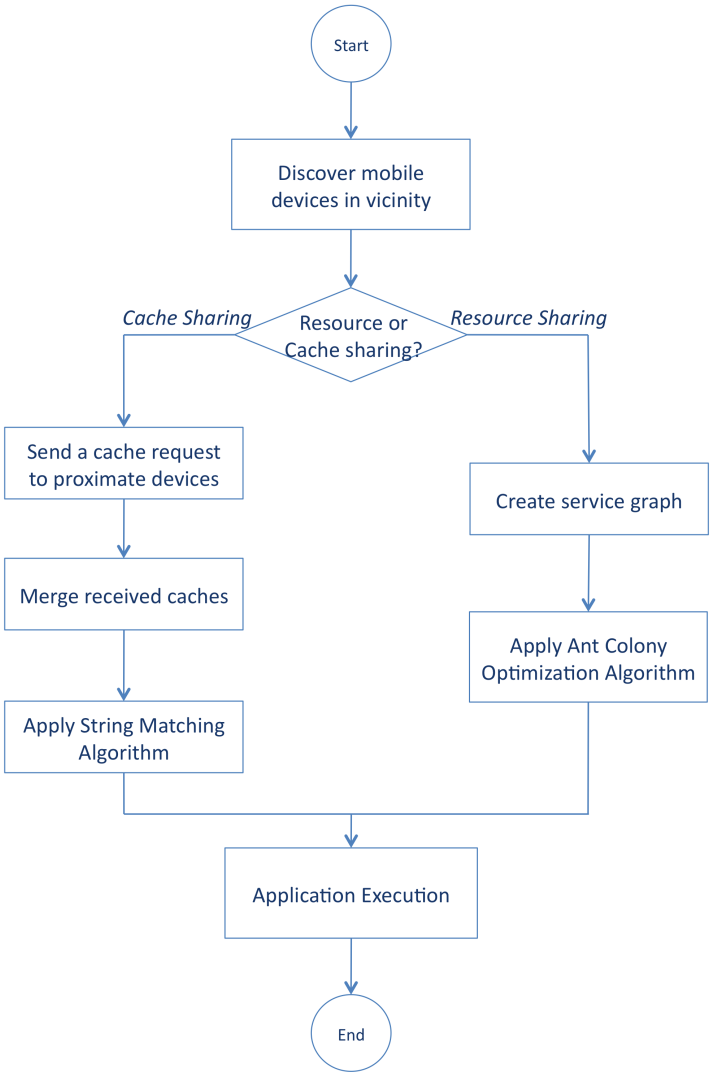

Figure 5.4: General flow of collaborative offloading

ACOMMA applies an ACO algorithm to determine which part of an application should execute on which mobile device. In the following, we explain these steps in detail.

5.2.1.1 Creating Service Graph for Multi Destination Application Offloading

Such as application offloading to the cloud that we explained in the previous chapter, for offloading onto several devices in the vicinity, the offloading decision making phase could be done using an application partitioning algorithm resulting to several partitions instead of two major parts for remote and local execution. The number of these partitions is at least two for executing on the

mobile device itself and up to the number of discovered nearby devices where the mobile device makes use of all neighbours for offloading. Figure 5.5 shows a sample graph portioning for offloading onto three nearby devices.

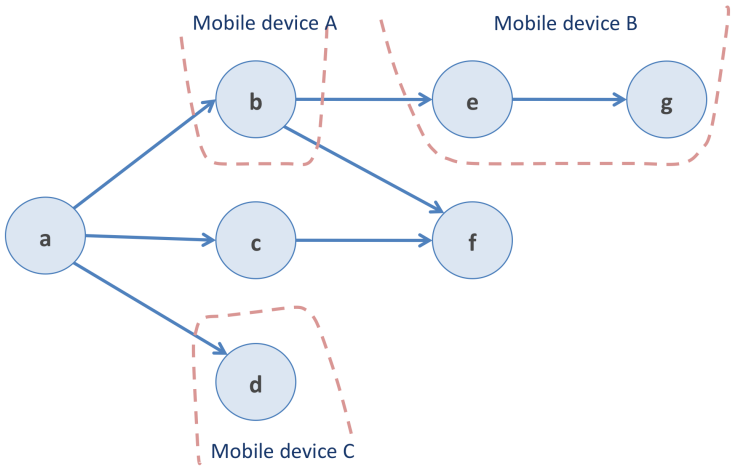

Figure 5.5: Application partitioning for multi destination offloading

Based on this partitioning, nodes *a, c, f* execute locally where node *b* executes on mobile device *A*, node *e and g* execute on mobile device *B* and finally mobile device *C* executes node *d.*

To be able to evaluate our proposed decision making algorithms also for multi destination offloading, we need to create an application service graph adaptable to the SP Problem. To this end, instead of duplicating each node of the call graph, ACOMMA adds for each method several nodes based on the number of discovered devices. In other words, it adds a new node to the graph for executing each method on each device. The edges between the nodes represent method invocations on the owner of that method. For instance in figure 5.6, for offloading onto mobile devices *A, B* and *C,* three nodes are created for *method 2* which is the only offloadable method in the original call graph.

Passing through *path1*, *path2* and *path3* makes the *method 2* to be executed on mobile device *A, B* and *C* respectively.

ACOMMA creates such a graph automatically at runtime after discovering nearby devices. The created graph remains static during one run of an application and is regenerated for the next executions. The following section describes how ACOMMA applies an ACO to find the shortest

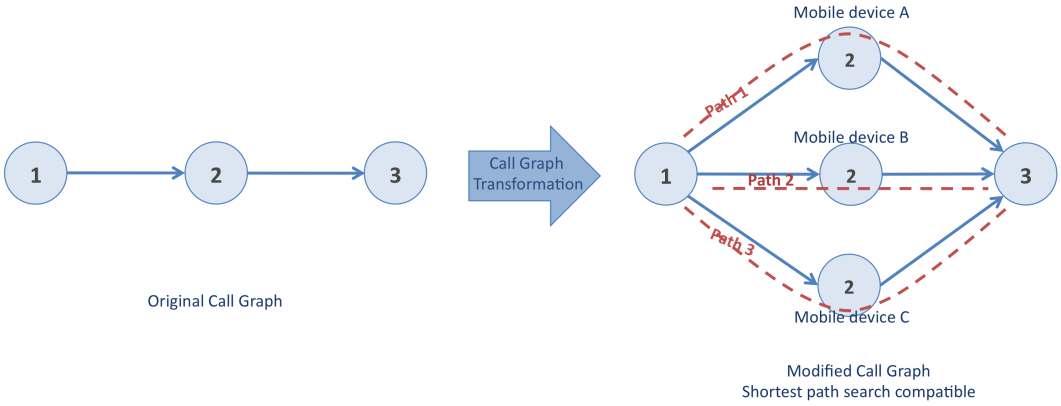

Figure 5.6: Call graph modification for multi destination offloading

path in this graph between the start and end points of the application.

5.2.1.2 Applying ACO for Multi Destination Decision Making

The ACO algorithm that we proposed for single destination offloading could be used also for multi destination offloading. In fact, this algorithm is not aware of the destinations and just searches for the shortest path with regard to the weights of the edges. These weights correspond to the execution time and CPU usage of a method execution on different devices. This algorithm is applicable for different graph sizes.

Whenever a method call happens, the offloading manager runs the ACO algorithm to decide where to execute this method. It randomly selects one of the paths from the non-dominated path set found by the optimization algorithm. The weights of the edges in the graph are all zero at initialization phase. After each method execution, the weight of the correspondent edges are updated with real values. These updates make the ACO algorithm result in more realistic non-dominated path set.

To make decisions that are adaptable to changing environments, ACOMMA runs the ACO algorithm on the total call graph for each method call. This helps the offloading manager to take into account environmental changes specially the disappearance of nearby devices while offloading. Although, the service graph remains static during an application run, several executions of the ACO during one application execution, make ACOMMA be notified about devices that are no

longer available. When the offloading manager sends an execution request to a device that already left the SPC, it will not receive any response before expected time out. So it updates the weight of the edge leading to a failed method on the left device node into a large value and re execute its optimization algorithm. In the new execution, the left device will not be selected because of the big weight of its related edges. Using the ACO, a mobile device could be aware of leaving devices during offloading even with its static graph, although it could not benefit from the new arriving devices during one application run.

In our proposed approach, the service graph size depends on the number of nearby devices. Any increase in the size of the SPC enhances the graph size and consequently more space is required for storing it. In addition, the bigger the graph size the more execution time of ACO algorithm which decrease ACOMMA's efficiency. Despite this limitation, we apply the ACO algorithm as a solution for the Shortest Path Problem for both single and multi destination offloading in order to be able to verify its efficiency in different situations. In addition, concerning the policy and privacy issues making mobile devices to collaborate is easier in domestic areas where the number of mobile devices and consequently the graph size is not so large that it would impose considerable overhead. A mobile device could benefit from both single and multiple destination offloadings depending on the context.

Overcoming this issue is possible by using a new approach such as task assignment or load sharing solutions; however, we could not find any evidence that shows the better performance of such algorithms compared with our proposed solution. Modifying the current approach could be also helpful, for instance we could consider some limitations for the destination devices in terms of their distance to offloading source or their total number. Another possible modification is making a two phase offloading decision making. In this case, the SPC is divided into smaller sub-clouds where, in the first offloading phase, the offloading manager decides for offloading to these small clouds instead of mobile devices, and in the second phase the header of each sub-cloud decides for execution of this method on one of its member devices. Using this two-phase decision making, mobile devices, in addition to share their resources, assume a part of the decision making cost. Another solution is making learning based decisions while devices collaborate in term of decision cache sharing. The usage of already made decisions prevents the reproduction of graphs and the execution of the ACO algorithm for every decision.

In the following, we explain how ACOMMA makes cache sharing between nearby mobile devices possible.

5.2.2 Collaboration-Based Decision Sharing in Application Offloading

The cooperation between mobile devices is used in many researches to achieve goals such as speech recognition [26], face detection and photography task [46], sharing Internet access [39] and data offloading [36].This cooperative working named collaborating or crowdsourcing.

We proposed mobile devices, in addition to share their resources and work load, to also share their decision cache that make collaborative decision making possible. This is an extension of the learning based decision making where mobile devices could apply the same decision as their neighbour in similar situation.

Although learning is one of the primary functions of dynamic systems such as sensor networks and mobile networks, it is mainly used for the establishment of a network with all its connections and the adaptation to the environment, but not for collaborative decision making. A handful of researches has been conducted on the usage of learning in offloading on a SPC. For example, [118] considers mobile devices as a series of experts temporarily coupled in a particular time and place in a way that the recent action of a mobile device influences its next state and the machines that are connected to it. Having a state, the experts available at the time should get together to pick the best available action. Therefore, online learning algorithms have been used in this framework.

Using ACOMMA, nearby mobile devices could share their decision's cache that includes the history of offloading decisions made for each application run. Sharing these caches is a type of training that makes mobile devices to know where to execute an application method in different situations without running its ACO Algorithm. The next chapter explains how the offloading manager of ACOMMA benefits from this information to offload application methods.

5.2.2.1 Collaborative Decision Sharing

ACOMMA makes use of the already introduced learning based decision making algorithm for mimetic collaborative decision making where the decision engine finds a solution in the cache using a SM algorithm. In fact, in both cases it decide to execute a method either locally on mobile device or remotely on a DS based on its cache. The important advantages of collaborative decision making

is benefiting from the experiences of the others while offloading. In a highly changing environment, it is helpful to have a richer cache for different situations. It may prevent to make several tries and errors in such a changing environment to find new solutions adapted to current changes.

Since mobile devices vary in terms of hardware characteristics and maybe contextual conditions and their decisions depend on their status at offloading time, adding contextual information to the cache could help to finding more adaptable decisions. For example, to take into account the amount of battery while offloading as well as the network status, we could define three different ranges for each parameters as follows:

Battery: *low-battery* , *medium-battery* , *high-battery*

Network speed: *low-speed*, *medium-speed*, *high-speed*

These intervals could have any predefined border.

The execution trail of the application in addition to this contextual information creates more realistic cache in which SM could find more adapted decisions. Figure 5.7 illustrates such a complete cache for making collaborative decisions using a SM algorithm.

1	*Low_speed medium_battery*	a_L	b_L	c_R	d_L	e_L
2	*high_speed low_battery*	a_L	b_R	c_R	d_R	e_L
3	*medium_speed high_battery*	a_L	b_R	c_L	d_R	e_L
4	*Low_speed high_battery*	a_L	b_L	c_L	d_L	e_L
5	*high_speed medium_battery*	a_L	b_R	c_R	d_R	e_R

Figure 5.7: A decision cache composed of decision trails and contextual information

While the same SM algorithm is used for both individual and collaborative learning based offloading decision making, the performance of collaborative decision making is extremely dependent on the cache.

How the cache gets filled, managed and emptied are the important points that we explain in the next section.

5.2.2.2 Decision Cache Management

Cache request

As the first step, the mobile device needs to access the cache of its neighbours assuming that, with the help of the discovery service, it already knows them. To this end, there are three different possible ways:

- On demand: In this method, a mobile device broadcasts a cache request to the nearby devices whenever it needs. In response, it will have cache if there is any in its neighbourhood.
- Periodically: In this method, each mobile device periodically sends its decision cache to its neighbours without concern to their requirement. They may keep or delete this information based on their needs.
- On changed: In this way, a mobile device sends its decision cache whenever it is modified either by adding a new execution trail or deleting old ones.

ACOMMA applies on changed cache broadcasting for decision sharing.

Cache merge

Depending on the number of devices in its vicinity a mobile device may receive several number of decision caches that should be merged to be ready for use for decision making. Some simple merging policies are listed here:

- Simple merge: in this way all received pieces of information are added to the local cache one by one, that may cause large local cache size related to the number of received data. This merging method is suitable for small groups of mobile devices.
- Unique merge: In this method, a local cache just keeps one copy of each application execution trail even if there exists several copies of it in the received data.
- Weighted merge: In this method, a local cache keeps one copy of each application execution trail while it attributing a weight that shows the number of times that the application executed this way. This may help the decision engine to learn more appropriate and useful execution trails while more occurrences means more efficiency of this selected execution trail.

- Categorized merge: Since the origin of this received cache are mobile devices with different hardware characteristics that made decisions in different contexts, the already taken decision may be kept in a categorized way that shows each execution trail category that is suitable for such a context. Using this categorized cache, the decision engine could start searching for a category that is more similar to the current situation of the mobile device

While testing a collaborative decision making with a few number of mobile devices, we apply simple merge for creating the local cache.

Cache invalidation

To avoid the cache size to augment exponentially we need to apply some invalidation policies to delete its data and keep it in a limited size. Some of these policies are as follows:

- Periodic invalidation: the cache is cleared with a predefined period of time using this invalidation policy

- On changed invalidation: since the received caches are those of the devices in the the current state of mobile device, while mobile device changes its place, start working between a new group of devices, so it is better to clear its cache and starts working with new information related to new neighbours.

- Categorized invalidation: it is possible that the mobile device keeps decisions that are more related to its current status, for example when having a fully charged battery, mobile devices could delete decisions taken in low battery situations since they may be less useful to lead to make efficient offloading decision.

Applying different combinations of these policies for cache management may greatly change the performance of ACOMMA for offloading using collaborative decision making.

In the next chapter, we evaluate ACOMMA for performing single and multiple destination offloading as well as individual and collaborative decision making.

Chapter 6

Implementation and Evaluation of ACOMMA

6.1 Validation Approach . . . 91
6.1.1 Applications . . . 91
6.1.2 Experimental Platform . . . 93
6.1.3 Success Criteria . . . 94
6.2 Evaluation . . . 94
6.2.1 Evaluation of Individual Decision Making for Single Destination Offloading 94
6.2.1.1 Ant Colony Optimization Performance . . . 95
6.2.1.2 String Matching Performance . . . 100
6.2.2 Evaluation of Collaborative Offloading . . . 101
6.2.2.1 Decision Sharing . . . 102
6.2.2.2 Resource Sharing . . . 104

This chapter starts with details of ACOMMA's implementation as well as its testbed and benchmarks and continues with ACOMMA's evaluation. We study the performance of ACOMMA in different scenarios for making single/collaborative offloading decisions to offload onto single/multiple destinations. The results of several tests and their analysis are presented in this chapter.

6.1 Validation Approach

6.1.1 Applications

To evaluate the design of ACOMMA, we started with four simple micro benchmarks and extend our tests with two macro benchmarks that are representative of popular applications. As micro

benchmarks, we developed simple mathematical functions as follows:

- Fibonacci sequence: It can be represented as shown below

$$a_0 = 1,\ a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2}$$

  This function gets an integer number as input and calculates its Fibonacci sequence.

- Determinant: It generally can be calculated by the following formula

$$det(M_{n*n}) = \sum_{i=0}^{n-1} Minor\ _{0,i}$$

$$Minor\ _{0,i} = (-1)^i * M_{0,i} * det(MinorMatrix\ _{0,i})$$

  where Minor matrix i, j is a matrix of size (n-1) which is a copy of matrix M where row i and column j are removed. This function gets the size of the matrix in terms of numbers of columns and rows and generates a random square matrix for calculating its Determinant.

- Integral function: It is calculated using the following formula

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{i=0}^{(b-a)/h} h * f(a + i * h)$$

  for the function of:

$$f(x) = \frac{1}{|cos(x) + sin(x)|}$$

  where the function input represents the intervals.

- Matrix Multiplication: It applies on two randomly generated square matrices in the following way

$$R = A * B$$

$$R_{i,j} = \sum_{k=0}^{n} A_{i,k} * B_{k,j}$$

  Same as Determinant, this function gets the size of matrix as input.

Although these functions are short and simple, they are different enough to allow us to do variety of tests in a first step. Fibonacci and matrix multiplications are both composed of a few number of methods but with different mathematical complexities. Fibonacci repeats a basic mathematical operation many times where matrix multiplications do some more complicated calculations. Determinant and integrate have some more number of methods that could be offloaded where the Determinant works recursively. Varying the inputs of each of these functions leads to interesting results.

We use macro benchmarks as examples of computation-intensive and interactive applications that are more currently used. We consider chess game and image processing applications and instead of a complete application, we implement their core algorithms. The number of offloadable methods in these benchmarks are much higher than for the micro benchmarks. Micro benchmarks are composed of 1 upto 5 methods while the average number of methods is 28 in macro benchmarks. The used algorithms are:

- The Monte Carlo algorithm is a randomized algorithm whose running time is deterministic, but whose output may be incorrect with a certain (typically small) probability [11]. This algorithm could be used for the choice of the next move in a chess game. Our developed algorithms make search in a tree and as input they get the number of nodes to start and the depth of search.

- The Face Recognition algorithm tries to match a given face image to a set of given face images using a number of eigenfaces [3] and is representative of image processing applications. The size of the input image in pixels and the number of searching images to match are the inputs of this algorithm.

These applications are all installed and run on Android smartphones and their service graph is created by ACOMMA at runtime.

6.1.2 Experimental Platform

We used a MacBook Pro with 8 GB of memory, a 250 GB hard disk and a 2,53 GHz Intel processor dual-core as our remote server. This server has OS X 10.9.5 Mavericks as operating system.

We used two different clients to evaluate the decision making process, the first one is a Samsung Galaxy SII with 1,2 GHz dual-core processor and 2GB of memory running Android version 4.1.2 (Jelly Bean). The second is Asus Google Nexus 7 Tablette with quad-core 1.2 GHz processor and 1GB of memory running Android version 5.1.1 (Lollipop).

6.1.3 Success Criteria

To successfully validate ACOMMA, we need to show the following things:

- ACOMMA is able to make a correct and efficient offloading decision to improve application performance by selecting an appropriate execution path on the application service graph using the ACO algorithm. It may also ameliorate its performance while making offloading decisions benefiting from its SM algorithm.

- ACOMMA is able to discover the neighbours of a mobile device and benefits from their decision cache to make offloading decisions.

- ACOMMA is able to correctly take multi destination offloading decisions to execute resource intensive application methods onto the nearby devices in an efficient way in the absence of server/cloud.

The validation of these three parts will justify our claim that ACOMMA is valuable.

6.2 Evaluation

6.2.1 Evaluation of Individual Decision Making for Single Destination Offloading

To evaluate the decision making process of ACOMMA, we ran several tests on each benchmark and we compared the total execution time and CPU usage of an application execution when it executes locally on mobile devices with its execution when offloaded by ACOMMA. To be able to compare the offloading gain in different execution complexities we ran each application with different inputs 25 times for each. Inputs of micro and macro benchmarks are listed in table 6.1.

	Fibonacci	Multiplication	Determinant	Integrate	Face Recognition	Monte Carlo
Serie 1	500	50x50	2	1.0-1.5	100000x1	10-5
Serie 2	1000	60x60	3	1.0-2.0	100000x2	20-7
Serie 3	1500	70x70	4	1.0-2.5	100000x3	30-9
Serie 4	2500	80x80	5	1.0-3.0	100000x4	40-11

Table 6.1: Test inputs for individual decision making using ACO

In the following, we evaluate the performance of ACOMMA for taking individual decisions using ACO and SM algorithms.

6.2.1.1 Ant Colony Optimization Performance

Since the execution time and CPU usage of application methods are decision making criteria that ACO uses for graph weights while solving the BSP problem, we show the gain in terms of these parameters while offloading.

Execution Time

Figure 6.1 shows the total execution time of local execution and offloading by ACO for different benchmarks running on Galaxy SII. The results of the four series of inputs are illustrated continuously.

As it shown in this figure, however Fibonacci and Matrix multiplications gains in terms of execution time using ACO, the gain of Determinant and Integral is much higher. Fibonacci and matrix multiplication use simple calculations that do not consume considerable resources. In addition their consumption growth rate is very small. So offloading is less efficient for them compared with more consuming applications and even in some runs, offloading execution take more time than local execution. Contrariwise, Integral and Matrix Determinant are consuming benchmarks with a considerable consumption growth rate as input changing peaks are visible on the figures. Using ACO the most consuming parts of the application execute on the server and while the execution time of these parts with different inputs on the server is almost the same, the total execution time using offloading is in a same range while the local execution time grows exponentially with more consuming inputs. In fact, the more the application is resource consuming the more we gain using offloading.

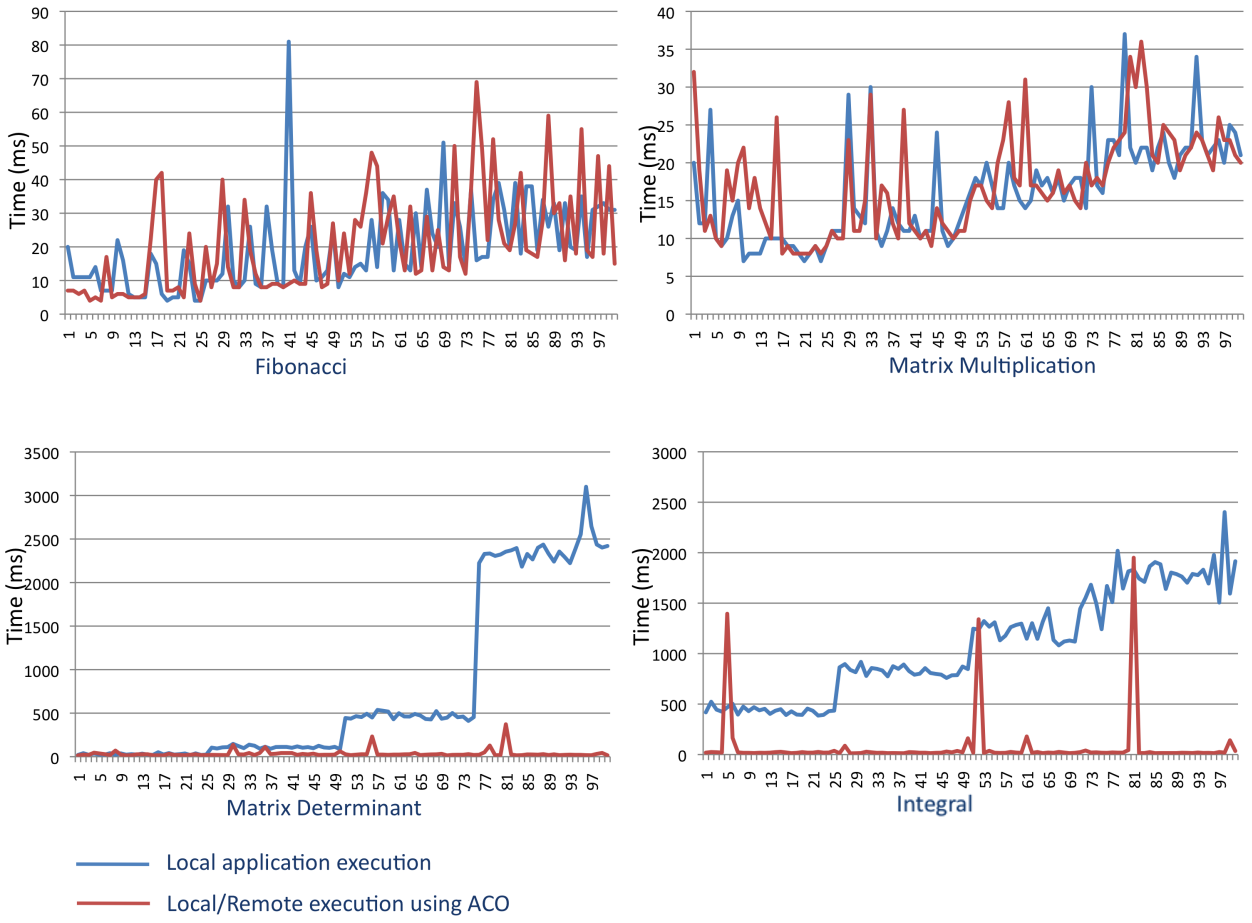

Figure 6.1: Local and ACO offloading execution time of micro benchmarks on Galaxy SII

The successful runs are the runs with their offloading execution time less than their local execution time, in other word, a run is successful if it gains in terms of execution time while offloading. The average success rate of Fibonacci and Matrix multiplication is 62% and %59.5 respectively while they augment to 94% and 96% for Determinant and Integral.

The peaks in the these figures where the red line exceeds the blue line show the unsuccessful test with much higher number of ACO executions than normal. The result of the same tests on Google Nexus 7 Tablette that are shown in figure 6.3 are also conform to the above mentioned results.

After testing ACO on micro benchmarks, we ran tests again for different inputs of Monte carlo and Face recognition algorithms again for 25 runs. Since these algorithms are resource-consuming

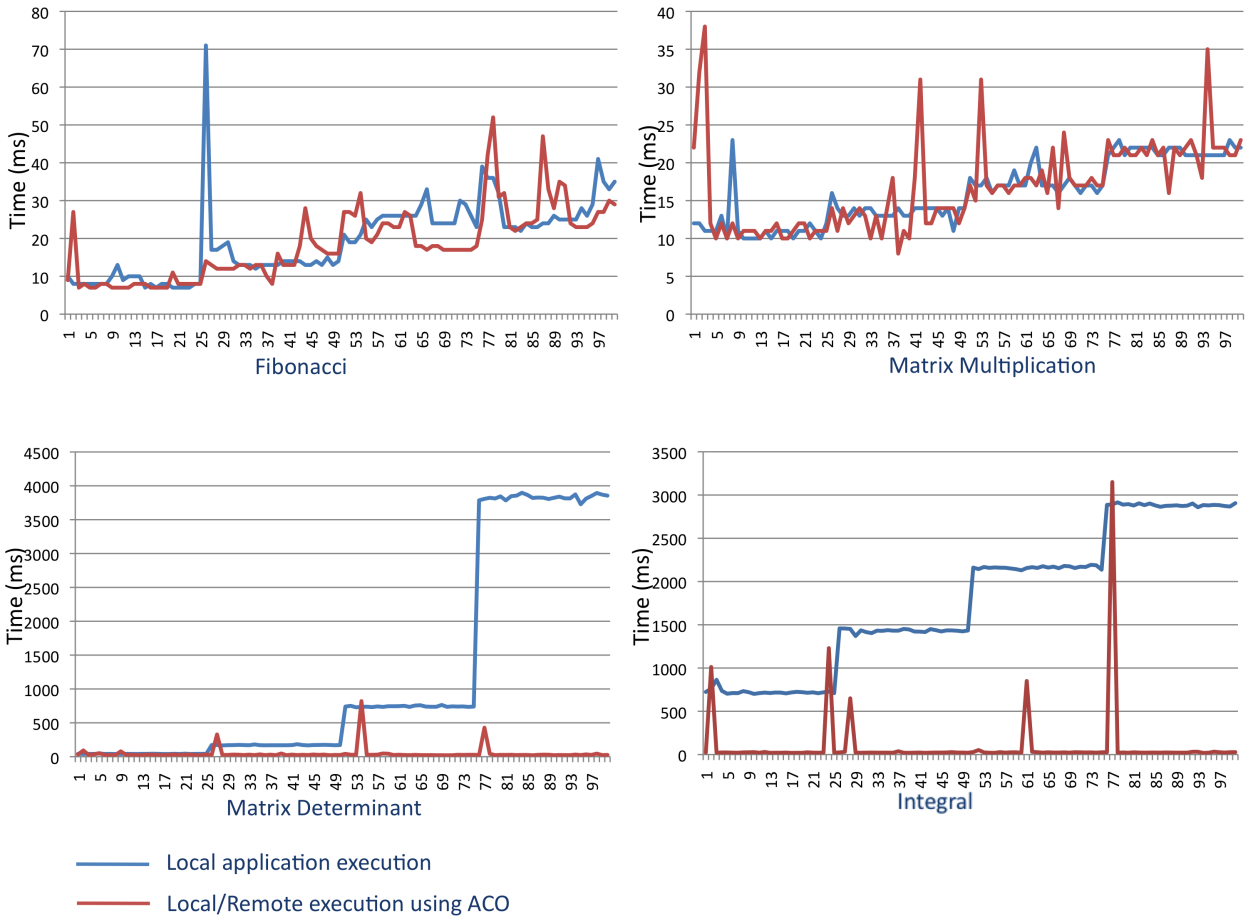

Figure 6.2: Local and ACO offloading execution time of micro benchmarks on Google Nexus 7 Tablette

ones, ACO works with an acceptable error rate for them as expected. The results of these executions on Galaxy SII is shown in figure 6.3.

The results show that the gain in terms of time for Face recognition and Monte carlo is less than for Integral and Determinant. These applications are all consuming but Face recognition and Monte carlo have a larger service graph that needs more time to find non dominated solutions. It seems that the efficiency of ACO depends on the graph complexity as well as the resource consumption of its nodes.

However ACO is not always efficient even for Integral and Determinant calculations and some offloading execution time is higher than the local execution time because ACO checks more paths to find a solution, it works well in general with a significant success rate. The time gained using ACO

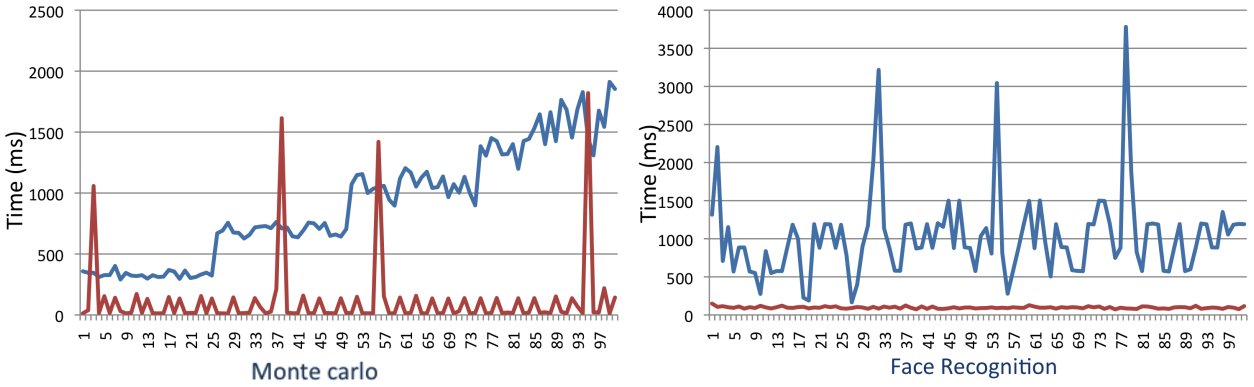

Figure 6.3: Local and ACO offloading execution time of macro benchmarks on Galaxy SII

is so great that ACOMMAS's overhead is negligible specially for more consuming runs. Figure 6.4 gives examples of ACOMMAS's overhead in terms of execution time compared with the time gained using ACO-based offloading for successful runs of Determinant and Integral.

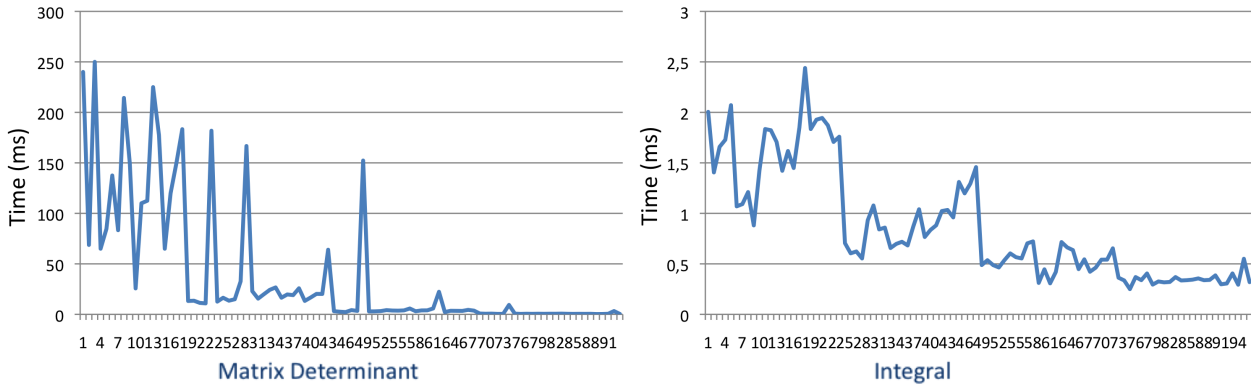

Figure 6.4: ACOMMA's overhead running ACO for successful runs of Determinant and Integral

CPU Usage

Coming to CPU usage as the second decision making criterion gives us almost the same results since ACO focuses on optimizing both of them. The unit of CPU usage measurement is the CPU cycle that is prepared by OS functions and is an integer number. So for methods that consume less than a cycle it maybe 0 or 1. On the same test runs, we evaluate the gain of CPU and again we have almost same results on both device models. Figures 6.5 is an example that shows the CPU usage of four micro benchmarks executed on Galaxy SII.

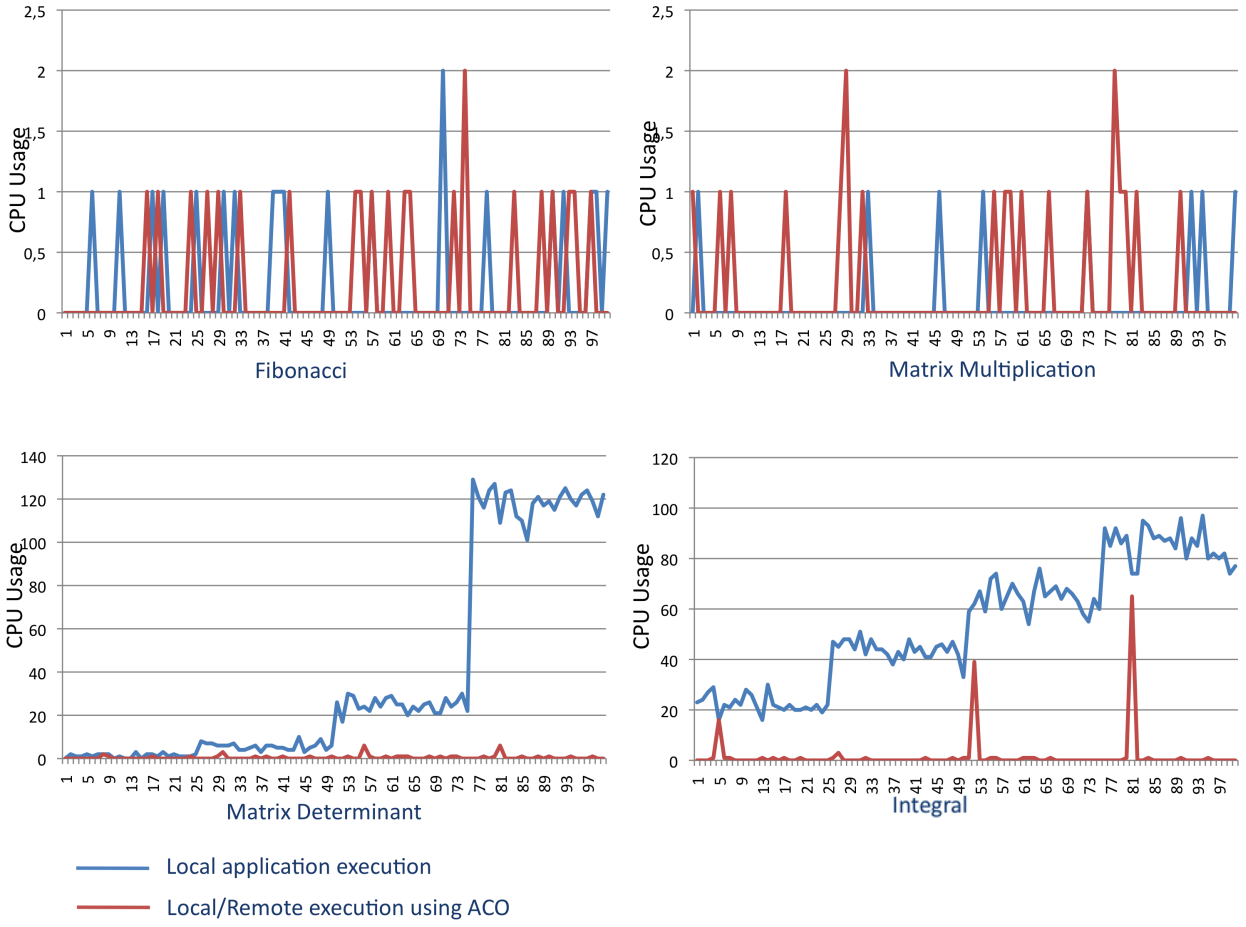

Figure 6.5: Local and ACO offloading CPU usage of micro benchmarks on Google Nexus 7 Tablette

Same as execution time evaluation, the ACO algorithm is less efficient for Fibonacci and Matrix multiplication compared with Determinant and Integral. Running the same tests on macro benchmarks result in the same conclusion.

As a summary of this section, we prepared tables that show the success rate, time gain and CPU gain of ACO while applying on different applications running on different devices. They also show the growth rate of the gain for larger inputs in Integral an Determinant. Table 6.3 and 6.2 are corresponding to the results of macro and micro benchmarks respectively.

Concerning these data we conclude that ACO is highly evaluated in gaining time and CPU usage with an average success rate of 77.87%.

		Fibonacci			Multiplication			Determinant			Integrate		
		Success (%)	Time (%)	CPU (%)	Success (%)	Time (%)	CPU (%)	Success (%)	Time (%)	CPU (%)	Success (%)	Time (%)	CPU (%)
Galaxy SII	Serie 1	60	40.54	20	48	7.6	-16.66	72	31.43	66.67	96	93.97	86.55
	Serie 2	64	29.7	18.75	60	11.93	0	100	66.20	93.73	96	96.99	99.49
	Serie 3	52	27.25	15.38	52	13.06	0	100	92.82	97.36	96	97.78	99.55
	Serie 4	56	31.08	7.14	52	8.37	15.38	100	98.23	99.54	96	98.59	99.81
Nexus 7 Tablette	Serie 1	76	13.34	10.53	48	9.74	-8.33	88	32.88	30.30	96	96.84	98.44
	Serie 2	52	18.98	23.08	44	7.03	0	96	83.78	82.97	96	98.39	98.65
	Serie 3	80	24.38	53.67	56	4.30	-7.14	96	96.23	98.53	96	98.85	99.18
	Serie 4	56	13.23	13.69	48	3.72	8.33	100	98.86	99.30	96	99.15	99.55

Table 6.2: Summary of individual decision making using ACO on micro benchmarks

		Face Recognition			MonteCarlo		
		Success (%)	Time (%)	CPU (%)	Success (%)	Time (%)	CPU (%)
Galaxy SII	Serie 1	100	83.92	81.43	96	81.66	99.74
	Serie 2	100	87.89	60.47	100	83.72	97.74
	Serie 3	100	88.52	75.23	96	94.68	99.92
	Serie 4	100	89.98	77.89	96	96.05	99.88
Nexus 7 Tablette	Serie 1	96	83.74	89.68	96	95.28	96.73
	Serie 2	92	82.09	91.71	100	94.39	97.36
	Serie 3	100	80.77	84.82	96	98.64	99.15
	Serie 4	96	78.40	75.42	96	99.01	99.34

Table 6.3: Summary of individual decision making using ACO on macro benchmarks

6.2.1.2 String Matching Performance

Although the overhead of ACO is negligible compared to its gain in terms of execution time, we apply a simple SM algorithm to verify if passing through the paths that are already determined by ACO in previous runs is beneficial. To this end, the already passed paths are saved in a cache. In the next runs, ACOMMA searches for matches in the cache firstly, if not found, it runs ACO. We used a naive SM function to find matches in the cache implemented in Java/Android.

For saving the paths in the cache we used a simple policy that just adds new paths at the end of the cache. ACOMMA searches a match in this cache and takes the first found one. We have tested SM without cache invalidation and with it. We applied periodically cache invalidation based on predefined run numbers.

We have tested SM on benchmarks whose performance ameliorates using the ACO algorithm. The average gain of SM with and without cache invalidation on both devices and for 100 runs, 25 for each series of inputs, is shown is table 6.4.

The results show that by using SM, the total execution time is only slightly improved. There is also no big differences between SM improvements with and without cache invalidation. They may happen for more complex applications with larger service graphs that imply a larger cache in terms

	Integrate		Determinant		Face Recognition		MonteCarlo	
	Without cache invalidation (%)	Periodic cache reset (%)	Without cache invalidation (%)	Periodic cache reset (%)	Without cache invalidation (%)	Periodic cache reset (%)	Without cache invalidation (%)	Periodic cache reset (%)
Galaxy S2	1.89	1.81	2.16	2.02	2.93	3.14	3.01	3.21
Tablette	1.72	1.80	1.83	1.79	2.91	2.89	2.86	2.91

Table 6.4: Execution time gained bye SM algorithm compared with ACO

of both path size and number of paths.

For periodic cache invalidation, we try 5 and 10 as run step to reset the cache and the results show that it does not change much. The results also show that for Face recognition and Monte carlo algorithms, SM works better than Integral and Matrix Determinant. We could conclude that SM is more adapted for the applications with more methods and more complicated service graphs so that ACO needs more time to evaluate a suitable path in it. In such complex applications, cache invalidation may also be more useful than for simple applications.

Based on the results shown in this section, we conclude that individual offloading decision making process of ACOMMA, either by ACO or SM algorithms, leads to a performance augmentation of resource constrained applications in terms of execution time and CPU usage, by offloading consuming methods onto a server.

6.2.2 Evaluation of Collaborative Offloading

In this section, we focus on evaluating ACOMMA when the mobile device is a member of a Spontaneous Proximity Cloud that collaborates with other devices for either resource or decision sharing. To this end as the first step, the mobile device should be able to discover its neighbours and communicate with them. We developed a decentralized discovery protocol using the services that Android offered. In this discovery method, all mobile devices register their services and port numbers related to them and then broadcast it. The device that needs to know its neighbours simply asks for received services and in this way it could start communication with nearby devices using the REST/HTTP protocol. Figure 6.6 shows a schema of 6 mobile devices that discover each other.

After the discovery phase, the mobile device could execute offloadable parts of an application on its neighbours or get their decision cache. In the following, we show the results of the collaboration phase.

Figure 6.6: Nearby device discovery

6.2.2.1 Decision Sharing

To share the decision cache between mobile devices, we apply a mimetic policy where each device sends its cache to its neighbours once it changes. The received cache is merged with the old one by deleting duplicate paths and taking their number as the path weight. The paths with higher weight have more chances to be selected by the SM algorithm.

We have tested decision sharing with 3 Galaxy SII and with Determinant, Integral, Face detection and Mont carlo as benchmarks. We have ran each benchmark 10 times and for two series of inputs as follows (table 6.5).

	Determinant	Integrate	Face Recognition	Monte Carlo
Serie 1	2	1.0-1.5	100000x1	10-5
Serie 2	3	1.0-2.0	100000x2	20-7

Table 6.5: Test inputs for collaborative decision sharing using ACO

In our scenario, a device has the role of source and two others are destinations. The source device runs an application in ACO-Collaboration mode that performs offloading using ACO algorithms

Benchmark	Determinant	Integrate	Face Recognition	MonteCarlo
Gain (%)	25.30	33.60	56.4	60.67

Table 6.6: Execution time gained by SM using collaborative cache compared with local cache

and saves discovered paths in its cache. Once its cache changed, it sends a copy of the cache to its neighbours. We ran an application 10 times on source devices resulting in a cache with a maximum size of 10 rows. When a source device has finished its execution, we ran the same application on destination devices in Collaboration mode that take decisions using the received cache and by the help of the SM algorithm.

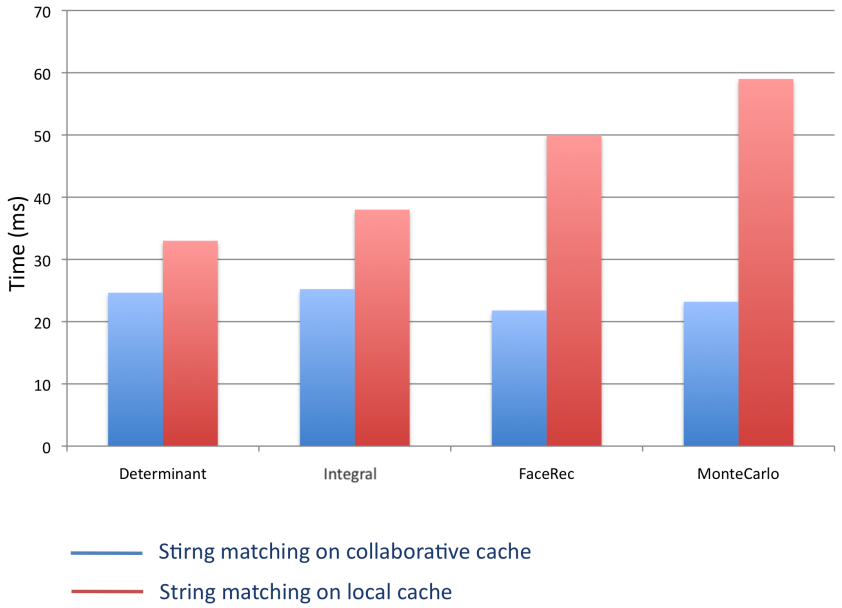

Figure 6.7: Execution time of application offloading, using SM by local or collaborative cache

The results show that applying SM for decision making based on collaborative cache is more efficient than making learning based decisions based on a local cache. Figure 6.7 shows a comparison of the execution time of the decision making process using local or collaborative cache. It seems that the gain in terms of time also depends on the graph size. Since Monte carlo implies more methods and a larger service graph as result, using a richer cache to find appropriate running solutions is more efficient for this benchmark. The table 6.6 shows how much an application gains in execution time while using collaborative cache compared with local cache.

Methods	Method A				Method B				Method C				Method D				Method E			
Offloading rate(%)	96 %				0 %				0 %				51 %				16 %			
Devices	d1	d2	d3	d4	d1	d2	d3	d4	d1	d2	d3	d4	d1	d2	d3	d4	d1	d2	d3	d4
Offloading portion(%)	32	29	23	16	0	0	0	0	0	0	0	0	36	31	18	15	28	29	26	17

Table 6.7: Execution trace of Determinant for multi destination offloading

Methods	Method A				Method B				Method C				Method D			
Offloading rate(%)	96 %				98 %				100 %				0 %			
Devices	d1	d2	d3	d4	d1	d2	d3	d4	d1	d2	d3	d4	d1	d2	d3	d4
Offloading portion(%)	36	28	18	18	27	26	24	23	26	26	30	18	0	0	0	0

Table 6.8: Execution trace of Integral for multi destination offloading

6.2.2.2 Resource Sharing

It is obvious that using the cloud as offloading destination is more efficient compared with offloading onto small mobile devices with limited resources. One of the main interests of using SPC for offloading is when the DS is not accessible for any reason. ACOMMA makes multi destination offloading possible using its ACO algorithm. In this section, we evaluate its dispatching between devices and make a comparison of the execution time for local execution, single destination and multi destination offloading. To evaluate its efficiency, we apply a scenario where 2 Galaxy SII and 3 Google Nexus 7 Tablettes create a SPC where a Galaxy SII makes offloading onto its neighbours while running Integral and Determinant as benchmarks. We used these benchmarks since ACOMMA works well for more consuming applications and the small number of their methods helps us to trace their execution.

As expected using SPC, offloading destination leads to less gain in terms of execution time as compared with offloading onto the cloud. The application execution time is also bigger than local execution time since the destination devices have the same or less processing power as the offloading source. In addition, the time of sending and receiving data is added to local execution time while offloading. The SPC is useful when a DC is not accessible and the mobile device has not enough resources for executing its applications. In this case, ACOMMA correctly selects destinations for offloading onto appropriate devices.

Tables 6.7 and 6.8 show the offloading trace for Determinant and Integrate respectively while using four nearby devices as destinations. In these tables, d1, d2, d3 and d4 are offloading destination devices where the first three are Google Nexsus Tablettes and the last one is Samsung Galaxy SII. The first raw shows how much a method is offloaded during 10 runs and the second raw shows the contribution of each device. For example as shown in table 6.7, method A gets offloaded in

96% of runs that between them the execution share of d1, d2, d3 and d4 are 32%, 29%, 23% and 16% respectively. It means that, for example, in 23% of runs, method A offloaded onto device 3. In all traces, the tablettes have almost the same portion of execution which is more than the portion of Galaxy SII and it is normal as all tablettes have the same hardware characteristics and are more powerful than Galaxy SII.

These results show that ACOMMA is greatly efficient to make multi destination offloading. In fact, based on all the above mentioned results, ACOMMA is evaluated as highly efficient to make dynamic offloading decisions using ACO and SM to offload onto single and multiple destinations.

Chapter 7

Conclusion

7.1 Summary

The work presented in this thesis focuses on mobile application augmentation by offloading on remote resources in the context of MCC. Current improvement in mobile device usage makes them more than a luxury but an essential part of human life that may connect every thing as well as users. The large range of the capabilities of these always connected devices in addition to their physical proximity to every day objects let them to be considered as go-between of users and their environmental objects that currently could think, feel, and talk by the virtue of embedded sensors. Although improvements in hardware platforms of mobile devices, make them almost powerful enough for supporting today's complicated and resource consuming applications, the battery is still a challenging point that maybe dealt with software solutions. In addition, the mobile device needs more resources as it is not only a personal device but also a communication port to the IoT. Trends to overcome the shortages of mobile devices from one hand, and the considerable success of CC on the other hand, lead to the emergence of MCC, a new paradigm that focuses on benefiting from cloud resources to overcome mobile device limitations through application offloading. In this scenario, the mobile device cloud be considered as a gateway that bridge IoT and CC.

In this thesis, we introduced the concept of SPC as a physically proximity cloud that could be used as an offloading destination instead of a DS whenever it is not accessible or very costly in terms of money or communication time. SPC is a self-construct/self-destroy cloud composed of nearby mobile devices that pool their resources and may join/leave the cloud irregularly.

Deciding for what, where and how to offload in highly changing environments is the main focus of any offloading middleware and several researches have tried to answer at least one of them.

In this thesis, we introduce ACOMMA, an Ant-inspired Collaborative Offloading Middleware for Mobile Applications that is equipped with several characteristics to deal with offloading challenges such as mobility and heterogeneity. Since mobile application offloading is a process between mobile device and the clouds, its challenges come out from both CC and MC as well as the communication challenges between them. Dealing with the heterogeneity of mobile devices, the open and service based architecture of ACOMMA makes it usable for any device that could support service communications using the standard protocol HTPP/REST. This open architecture also permits the mobile device to communicate easily with the IoT and SPC without any special API requirement. In such a service based architecture, a mobile application is modelled as a service graph where its nodes and vertices represent services and service invocations respectively. This graph is generated by ACOMMA itself and there is no need for user intervention for code modification or the addition of annotations.

We consider an application partitioning algorithm to determine the offloadable parts of an application where the service graph breaks apart into two or several partitions that represent different execution platforms. To make more adaptive offloading decisions, we proposed a fine-grained method-level application offloading. The fine granularity also makes it possible to offload onto SPC as well as a DS while these tiny partitions are adapted to be executed on resource constrained mobile devices in the vicinity.

In order to make adaptive offloading decisions in highly changing environments, we proposed bio-inspired algorithms and specially Bi-Objective ACO. Bi-objective decision making process of ACOMMA makes this dynamic decision more adapted to the mobile context. Execution time and CPU usage are the criteria that ACOMMA takes into account as decision making parameters that also cover network speed and battery implicitly. The ACO algorithm searches for the SP between the start and end points of the mobile application in the service graph where the selected nodes show where to execute the corresponding service. In order to react appropriately to a changing environment at runtime, ACOMMA applies ACO algorithm for each method call on the entire application graph.

ACOMMA also supports multi destination offloading that makes the collaboration of mobile

devices possible, while profiting from the resources of one another through offloading. By the help of its discovery service, ACOMMA gets acquainted with its nearby devices and applies the same ACO algorithm onto a modified version of the service graph to find the SP that determines where to execute offloadable nodes among several devices.

To avoid running ACO for any application offloading request, we proposed a learning based decision making algorithm that searches an already taken decision in similar situation in previous execution trails and applies it on the current offloading process. To this end, ACOMMA saves the execution trail of each run in a decision cache and uses a SM algorithm to search into this cache if required. This learning based decision making could be done in a collaborative way while nearby devices cooperate to create a collaborative cache which could be richer than a local one. In this collaborative decision making process, ACOMMA makes mimetic decisions based on the collaborative cache.

To evaluate ACOMMA, we developed a prototype and tested offloading efficiency for different applications and different entries. The results show that ACOMMA works greatly for single destination offloading of the applications that are more consuming and augment greatly application performance in terms of execution time and CPU usage. The more the application is consuming the more performance is augmented. This gain is high enough to make the overhead of ACOMMA negligible. The results also show that ACOMMA dispatches correctly the offloadable parts between SPC members based on their processing power while doing multi destination offloading. Applying SM algorithm to make learning based decisions is also slightly ameliorating the performance compared with ACO.

7.2 Short and Long Term Perspectives

Among our perspectives in the short and medium terms, we are interested in enlarging our evaluation with offloading more complex applications and specially multi-threads ones using ACOMMA and with more learning based decision making policies taking into account the context.

Although the results show that ACOMMA works greatly for more complex and more consuming applications, we do not have any threshold for this augmentation. Since the size of the call graph grows for bigger applications and in consequent there is much more paths to be traversed by ants, there maybe a limit above while the overhead of the decision making process dominates the

offloading gain. Supporting multi thread offloading is also a feature that we are interested in adding to ACOMMA to make simultaneous offloadings of several methods to single/multiple destination(s) possible.

While offloading more complex applications with a large service graph, there is a chance that performance augments much more using SM algorithm as traversing a large graph takes much more time than searching in a cache. In this case, there is much more paths with larger size in the cache emphasizing the importance of advanced policies for cache invalidation as well as cache merging for collaborative cache creation. Considering the context while taking learning based offloading decisions may also improve the performance of learning based decision making.

We are also interested in changing decision making criteria and also evaluate ACOMMA's performance in terms of energy consumption.

Coming to collaborative multi destination offloading we would like to test with bigger SPC with more devices to find the threshold of ACOMMA's success in this scenario.

As mobile devices are going to be more and more powerful even in terms of battery lifetime, the application offloading takes importance where mobile device plays the role of IoT gateway. The first important long term perspective of our work is testing ACOMMA while offloading received data / requests of daily object in ambient intelligence onto other nearby devices or a remote cloud. The way that it communicates with the objects and manages the offloading process to make profit of every things as well as itself through offloading are the challenging points of this idea.

Coming to collaboration of mobile devices and also of objects, the mobility may cause problems as there is no guarantee that a device stays available until the end of its dedicated task and it is more important when a mobile device takes the responsibility of offloading a group of objects. Dealing with this issue by simulating movement patterns of devices and making an offloading hand over is another long term perspective of this work. This is more feasible in small and closed spaces with a limited number of devices and objects.

In public areas, we place our interest on making mobile devices to collaborate to decision sharing for common parts of different applications. As different applications could have some common features so that their previous execution trail could be shared to help others while making offloading decisions, there may be some common applications that are used in different ways by different users. We are interested in taking into account these differences while making collaborative cache as well

as implementing collaborative decision making by applying algorithms such as consensus. We are also interested in applying some incentive algorithms to motivate users to accept collaboration in public areas.

Publications

National Conferences

- R. Golchay, F. Le Mouël, J. Ponge, and N. Stouls. Les smartphones comme passerelle de services: peuvent-ils relier l'Internet des choses (IoT) et la virtualisation dans les nuages (Cloud)? In Actes de la Conférence d'informatique en Parallélisme, Architecture et Systèmes (ComPAS'2013) - Conférence Française en Systèmes d'Exploitation (CFSE'9), Grenoble, France, January 2013.

- R. Golchay, F. Le Mouël, S. Frénot and J. Ponge. Towards Bridging IoT and Cloud Services: Enabling Smartphones as Mobile and Autonomic Service Gateways, In Actes des 7ème Journées Francophones de la Mobilité et Ubiquité (UbiMob'2011), pp.45-48, Toulouse, France, June 2011.

Research Report

- R. Golchay, F. Le Mouël, J. Ponge and N. Stouls. From Mobile Cloud Computing to Tangible Cloud Computing: A Survey, Research Report INRIA CITI Lab, INSA Lyon, 2013.

Bibliography

[1] Amazon elastic compute cloud (ec2). Available:http://www.amazon.com/ec2/. [Online; Accessed December 10th, 2011.].

[2] Dropbox. Available:https://www.dropbox.com/fr/.

[3] Face recognition algorithm. Available:https://code.google.com/p/javafaces/.

[4] Facebook. Available:http://facebook.com. [Online; Accessed November 26th, 2011.].

[5] Facebook for business. Available:https://www.facebook.com/business/a/local-awareness-ads.

[6] Flickr. Available:http://flickr.com. [Online; Accessed December 26th, 2011.].

[7] Google app engine. Available:http://appengine.google.com. [Online; Accessed 15 November 2011].

[8] Google for mobile. Available:http://www.google.com/mobile/mail/.

[9] Hadoop. Available:https://hadoop.apache.org/.

[10] Microsoft azure. Available:http://www.windowsazure.com. [Online; Accessed April 10th, 2012.].

[11] Monte carlo algorithm. Available:https://en.wikipedia.org/wiki/MonteCarloalgorithm.

[12] Opennebulu. Available:http://http://opennebula.org/.

[13] Twitter. Available:https://twitter.com/twittermobile.

[14] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *Communications Surveys Tutorials, IEEE*, 16(1):337–368, First 2014.

[15] Kassahun Adem, Caspar Ryan, and Ermyas Abebe. Crowdsourcing the cloud: Energy-aware computational offloading for pervasive community-based cloud computing. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 415. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.

[16] Tomi Ahonen. Household penetration rates for technology across the digital divide, 2011.

[17] Pelin Angin and Bharat Bhargava. An agent-based optimization framework for mobile-cloud computing. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(2):1 – 17, 2013.

[18] Jo ao Sousa and David Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43, Montreal, Canada, August 2002.

[19] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 87–92, New York, NY, USA, 2002. ACM.

[20] Rajesh Krishna Balan. *Simplifying cyber foraging.* School of Computer Science, Carnegie Mellon University, 2006.

[21] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 273–286, New York, NY, USA, 2003. ACM.

[22] Soumya Banerjee, Indrajit Mukherjee, and P Mahanti. Cloud computing initiative using modified ant colony framework. *World academy of science, engineering and technology*, 56:221–224, 2009.

[23] Roberto Beraldi, Khalil Massri, Abderrahmen Mtibaa, and Hussein Alnuweiri. Towards automating mobile cloud computing offloading decisions: An experimental approach.

[24] R. Bialek, E. Jul, J.-G. Schneider, and Yan Jin. Partitioning of java applications to support dynamic updates. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 616–623, Nov 2004.

[25] S Binitha and S Siva Sathya. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.

[26] Yu-Shuo Chang and Shih-Hao Hung. Developing collaborative applications with mobile cloud-a case study of speech recognition. *Journal of Internet Services and Information Security (JISIS)*, 1(1):18–36, 2011.

[27] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a good idea? In Jan Vitek and Christian Tschudin, editors, *Mobile Object Systems Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*, pages 25–45. Springer Berlin Heidelberg, 1997.

[28] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys '11*, pages 301–314, New York, NY, USA, 2011. ACM.

[29] Byung-Gon Chun and Petros Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 7:1–7:5, New York, NY, USA, 2010. ACM.

[30] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html`, 2014. [Online; accessed February 5, 2014].

[31] João Manuel Coutinho-Rodrigues, João CN Clímaco, and John R Current. An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions. 1999.

[32] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making smartphones last longer with code offload. In *Proc. of MobiSys '10*, pages 49–62, New York, NY, USA, 2010. ACM.

[33] Y. Cao D. Kovachev and R. Klamma. Mobile cloud computing: a comparison of application models. *arXiv preprint arXiv:1107.4940*, 2011.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, jan 2008.

[35] S. Deng, L. Huang, J. Taheri, and A.Y. Zomaya. Computation offloading for service workflow in mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2014.

[36] Aaron Yi Ding, Bo Han, Yu Xiao, Pan Hui, Aravind Srinivasan, Markku Kojo, and Sasu Tarkoma. Enabling energy-aware collaborative mobile data offloading for smartphones. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*, pages 487–495. IEEE, 2013.

[37] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.

[38] Manish Dixit, Nikita Upadhyay, and Sanjay Silakari. An exhaustive survey on nature inspired optimization algorithms. *International Journal of Software Engineering & Its Applications*, 9(4), 2015.

[39] Ngoc Do, Cheng-Hsin Hsu, and Nalini Venkatasubramanian. Crowdmac: a crowdsourcing system for mobile access. In *Proceedings of the 13th International Middleware Conference*, pages 1–20. Springer-Verlag New York, Inc., 2012.

[40] Karl Doerner, WalterJ. Gutjahr, RichardF. Hartl, Christine Strauss, and Christian Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 131(1-4):79–99, 2004.

[41] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, Apr 1997.

[42] Zuochao Dou. Benefits of Utilizing an Edge Server(Cloudlet) in the MOCHA Architecture. Master's thesis, SUniversity of Rochester,Rochester, New York, 2013.

[43] C. Doukas, Thomas Pliakas, and I. Maglogiannis. Mobile healthcare information management utilizing cloud computing and android os. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 1037–1040, Aug 2010.

[44] Dave Durkee. Why cloud computing will never be free. *Commun. ACM*, 53(5):62–69, May 2010.

[45] Afnan Fahim, Abderrahmen Mtibaa, and Khaled A. Harras. Making the case for computational offloading in mobile device clouds. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 203–205, New York, NY, USA, 2013. ACM.

[46] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Honeybee: A programming framework for mobile crowd computing. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 224–236. Springer, 2013.

[47] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[48] R. Ferzli and I. Khalife. Mobile cloud computing educational tool for image/video processing algorithms. In *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, pages 529–533, Jan 2011.

[49] Jason Flinn, SoYoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 217–226, 2002.

[50] Huber Flores and Satish Srirama. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In *Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services*, MCS '13, pages 9–16, New York, NY, USA, 2013. ACM.

[51] R. Frederking and R. D Brown. The pangloss-lite machine translation system. In *2nd Conf. of the Assoc. for Mach. Trans. in the Americas*, pages 268–272, 1996.

[52] Bo Gao, Ligang He, Limin Liu, Kenli Li, and S.A. Jarvis. From mobiles to clouds: Developing energy-aware offloading strategies for workflows. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 139–146, Sept 2012.

[53] Keivan Ghoseiri and Behnam Nadjari. An ant colony optimization algorithm for the bi-objective shortest path problem. *Appl. Soft Comput.*, 10(4):1237–1246, September 2010.

[54] Ioana Giurgiu, Oriana Riva, and Gustavo Alonso. Dynamic software deployment from clouds to mobile devices. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 394–414, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[55] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proc. of Middleware '09*, volume 5896 of *LNCS*, pages 83–102. Springer Berlin / Heidelberg, 2009.

[56] Michel Goraczko, Jie Liu, Dimitrios Lymberopoulos, Slobodan Matic, Bodhi Priyantha, and Feng Zhao. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 191–196, New York, NY, USA, 2008. ACM.

[57] Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, Z. Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 93–106, Berkeley, CA, USA, 2012. USENIX Association.

[58] Hatem Hamad, Motaz Saad, and Ramzi Abed. Performance evaluation of restful web services for mobile devices. *Int. Arab J. e-Technol.*, 1(3):72–78, 2010.

[59] Dijiang Huang, Xinwen Zhang, Myong Kang, and Jim Luo. Mobicloud: Building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 27–34, June 2010.

[60] Gonzalo Huerta-Canepa and Dongman 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*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM.

[61] Noha Ibrahim. *Spontaneous Integration of Services in Pervasive Environments*. PhD thesis, National Institude on Applied Sciences of Lyon, 2008.

[62] Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, CarlosArtemio Coello Coello, and David Corne, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 359–372. Springer Berlin Heidelberg, 2001.

[63] Saeed Javanmardi, Mohammad Shojafar, Danilo Amendola, Nicola Cordeschi, Hongbo Liu, and Ajith Abraham. Hybrid job scheduling algorithm for cloud computing environment. In Pavel KÃ¶mer, Ajith Abraham, and VÃ¡clav SnÃ¡Å¡el, editors, *Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014*, volume 303 of *Advances in Intelligent Systems and Computing*, pages 43–52. Springer International Publishing, 2014.

[64] P Jones. Industry census 2012: Emerging data center markets, 2012.

[65] Frédéric Le Mouël Julien Ponge. Jooflux: Hijacking java 7 invokedynamic to support live code modifications. Technical report, Technical report, CITI - Centre of Innovation in Telecommunications and Integration of services, 2012.

[66] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency. Technical report, Technical report, McKinsey & Company, 2008.

[67] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. Bal. eyedentify: Multimedia cyber foraging from a smartphone. In *Multimedia, 2009. ISM '09. 11th IEEE International Symposium on*, pages 392–399, Dec 2009.

[68] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A computation offloading framework for smartphones. In Martin Gris and Guang Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg, 2012.

[69] A.R. Khan, M. Othman, S.A. Madani, and S.U. Khan. A survey of mobile cloud computing application models. *Communications Surveys Tutorials, IEEE*, 16(1):393–413, First 2014.

[70] Olga Kharif. Average household has 5 connected devices, while some have 15-plus. *Bloomberg: Tech Blog, August*, 29, 2012.

[71] Lee Kilho and Shin Insik. User mobility model based computation offloading decision for mobile cloud. *Journal of Computing Science and Engineering*, 9(3):155–162, 2015.

[72] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Trans. Comput. Syst.*, 10(1):3–25, February 1992.

[73] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953, March 2012.

[74] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.*, 18(1):129–140, February 2013.

[75] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '01, pages 238–246, New York, NY, USA, 2001. ACM.

[76] Zhiyuan Li and Rong Xu. Energy impact of secure computation on a handheld device. In *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pages 109–117, Nov 2002.

[77] Nikitas Liogkas, Blair MacIntyre, Elizabeth D Mynatt, Yannis Smaragdakis, Eli Tilevich, and Stephen Voida. Automatic partitioning: Prototyping ubiquitous-computing applications. *IEEE Pervasive Computing*, (3):40–47, 2004.

[78] Jieyao Liu, Ejaz Ahmed, Muhammad Shiraz, Abdullah Gani, Rajkumar Buyya, and Ahsan Qureshi. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99 – 117, 2015.

[79] A. Colorni M. Dorigo, V. Maniezzo. Ant system: An autocatalytic optimizing process,. Technical report, Technical Report, Dipartimento di Elettronica e Informazione, Politec-nico di Milano,, 1991.

[80] Chi Harold Liu M. Reza Rahimi, Jian Ren, Athanasios V. Vasilakos, and Nalini Venkatasubramanian. Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, 19(2):133–143, 04 2014.

[81] Rakesh Madivi and S Sowmya Kamath. An hybrid bio-inspired task scheduling algorithm in cloud environment. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–7, July 2014.

[82] Rashmi A Mahale and SD Chavan. A survey: evolutionary and swarm based bio-inspired optimization algorithms. *International Journal of Scientific and Research Publications*, 2(12):1–6, 2012.

[83] Eugene E. Marinelli. *Hyrax: Cloud Computing on Mobile Devices using MapReduce*. PhD thesis, School of Computer Science Carnegie Mellon University Pittsburgh, 2009.

[84] Chonglei Mei, D. Taylor, Chenyu Wang, A. Chandra, and J. Weissman. Sharing-aware cloud-based mobile outsourcing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 408–415, June 2012.

[85] Emiliano Miluzzo, Ramón Cáceres, and Yih-Farn Chen. Vision: Mclouds - computing on clouds of mobile devices. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, pages 9–14, New York, NY, USA, 2012. ACM.

[86] Jayadev Gyani.P.R.K.Murti M.Rajendra Prasad. Mobile cloud computing: Implications and challenges. *Journal of Information Engineering and Applications*, 2(7), 2012.

[87] A. Mtibaa, M. Abu Snober, A. Carelli, R. Beraldi, and H. Alnuweiri. Collaborative mobile-to-mobile computation offloading. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pages 460–465, Oct 2014.

[88] Abderrahmen Mtibaa, Afnan Fahim, Khaled A. Harras, and Mostafa H. Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. *SIGCOMM Comput. Commun. Rev.*, 43(4):51–56, August 2013.

[89] ABI Research News. More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020. `https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne`, 2013. [Online; accessed May 09, 2013].

[90] Gartner Newsroom. Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020. `http://www.gartner.com/newsroom/id/2636073`, 2013. [Online; accessed December 12, 2013].

[91] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden. Wishbone: Profile-based partitioning for sensornet applications. In *Proc. of NSDI '09*, pages 395–408, Berkeley, CA, USA, 2009. USENIX Association.

[92] Ralf Niemann and Peter Marwedel. An algorithm for hardware/software partitioning using mixed integer linear programming. *Des. Autom. Embedded Syst.*, 2(2):165–193, March 1997.

[93] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. *SIGOPS Oper. Syst. Rev.*, 31(5):276–287, October 1997.

[94] Gabriel Orsini, Dirk Bade, and Winfried Lamersdorf. Context-aware computation offloading for mobile cloud computing: Requirements analysis, survey and design guideline. *Procedia Computer Science*, 56:10 – 17, 2015. The 10th International Conference on Future Networks and Communications (FNC 2015) / The 12th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2015) Affiliated Workshops.

[95] Mazliza Othman and Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):44–51, January 1998.

[96] V.S. Pendyala and J. Holliday. Performing intelligent mobile searches in the cloud using semantic technologies. In *Granular Computing (GrC), 2010 IEEE International Conference on*, pages 381–386, Aug 2010.

[97] T. Penner, A. Johnson, B. Van Slyke, M. Guirguis, and Qijun Gu. Transient clouds: Assignment and collaborative execution of tasks on mobile devices. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2801–2806, Dec 2014.

[98] Shanmuga Sundaram Senthil Rajan Kesavan Periyar Dasan, Agan Prabhu. A ubiquitous home control and monitoring system using android based smart phone for iot. *International Journal of Computer Science and Mobile Computing*, 2(12):188–197, 2013.

[99] Padmanabhan S. Pillai, Lily B. Mummert, Steven W. Schlosser, Rahul Sukthankar, and Casey J. Helfrich. Slipstream: Scalable low-latency interactive perception on streaming data. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '09, pages 43–48, New York, NY, USA, 2009. ACM.

[100] Han Qi and A. Gani. Research on mobile cloud computing: Review, trend and perspectives. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pages 195–202, May 2012.

[101] S.S. Qureshi, T. Ahmad, K. Rafique, and Shuja ul islam. Mobile cloud computing as future for mobile applications - implementation methods and challenging issues. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 467–471, Sept 2011.

[102] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: Enabling interactive perception applications on mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[103] Amol B. Bhande Rahul B. Mannade. Challenges of mobile computing: An overview. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(8):3109–3114, 2013.

[104] Amol B. Bhande Rahul B. Mannade. Challenges of mobile computing: An overview. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(8), Augest 2013.

[105] Jan S. Rellermeyer, Oriana Riva, and Gustavo Alonso. Alfredo: An architecture for flexible interaction with electronic devices. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 22–41, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[106] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):19–26, January 1998.

[107] Abdullah Gani Saeid Abolfazli, Zohreh Sanaei. Mobile cloud computing: A review on smartphone augmentation approaches. In *International Conference on Computing, Information Systems, and Communications(CISCO'12), Singapore*, May 2012.

[108] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 1–7, New York, NY, USA, 1996. ACM.

[109] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, Aug 2001.

[110] Mahadev Satyanarayanan, P. Bahl, R Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, Oct 2009.

[111] Markus SchÃ$\frac{1}{4}$ring. Mobile cloud computing-open issues and solutions, 2011.

[112] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys Tutorials, IEEE*, 14(4):1232–1243, Fourth 2012.

[113] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 145–154, New York, NY, USA, 2012. ACM.

[114] M. Shiraz, A. Gani, R.H. Khokhar, and R. Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys Tutorials, IEEE*, 15(3):1294–1313, Third 2013.

[115] Kanad Sinha and Milind 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*, CCGRID '11, pages 184–194, Washington, DC, USA, 2011. IEEE Computer Society.

[116] Krzysztof Socha, Michael Sampels, and Max Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In Stefano Cagnoni, ColinG. Johnson, JuanJ.Romero Cardalda, Elena Marchiori, DavidW. Corne, Jean-Arcady Meyer, Jens Gottlieb, Martin Middendorf, AgnÃ¨s Guillot, GÃ$\frac{1}{4}$ntherR. Raidl, and Emma Hart, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. Springer Berlin Heidelberg, 2003.

[117] Wei-Tse Tang, Chiu-Ming Hu, and Chien-Yeh Hsu. A mobile phone based homecare management system on the cloud. In *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on*, volume 6, pages 2442–2445, Oct 2010.

[118] Cem Tekin and Mihaela van der Schaar. An experts learning approach to mobile service offloading. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 643–650. IEEE, 2014.

[119] Eli Tilevich and Yannis Smaragdakis. J-orchestra: Enhancing java programs with distribution capabilities. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(1):1, 2009.

[120] A. Tuli, N. Hasteer, M. Sharma, and A. Bansal. Exploring challenges in mobile cloud computing: An overview. In *Confluence 2013: The Next Generation Information Technology Summit (4th International Conference)*, pages 496–501, Sept 2013.

[121] A. Rasool V. SaiKrishna and N. Khare. String matching and its applications in diversified fields. *International Journal of Computer Science Issues*, 9(1):219–226, 2012.

[122] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesińska, Rutger F. H. Hofman, Ceriel J. H. Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: A flexible and efficient java-based grid programming environment: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(7-8):1079–1107, June 2005.

[123] Tim Verbelen, Tim Stevens, Filip De Turck, and Bart Dhoedt. Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. *Future Gener. Comput. Syst.*, 29(2):451–459, February 2013.

[124] Alex Waibel. Interactive translation of conversational speech. *Computer*, 29(7):41–48, Jul 1996.

[125] Cheng Wang and Zhiyuan Li. Parametric analysis for adaptive computation offloading. *SIGPLAN Not.*, 39(6):119–130, June 2004.

[126] Yating Wang, Ing-Ray Chen, and Ding-Chau Wang. A survey of mobile cloud computing applications: Perspectives and challenges. *Wirel. Pers. Commun.*, 80(4):1607–1623, February 2015.

[127] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. Using bandwidth data to make computation offloading decisions. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[128] Yejin Shin Wooseong Kim and Soonuk Seol. Smart phone assisted personal iot service. *Advanced Science and Technology Letters*, 110:61–66, 2015.

[129] Lei Yang, Jiannong Cao, and Hui Cheng. Resource constrained multi-user computation partitioning for interactive mobile cloud applications. Technical report, Technical report, Department of Computing, Hong Kong Polytechnical University, 2012.

[130] Lei Yang, Jiannong Cao, Shaojie Tang, Tao Li, and A.T.S Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 794–802, June 2012.

[131] Xiaoyan Yang, Tiejun Pan, and Jingjing Shen. On 3g mobile e-commerce platform based on cloud computing. In *Ubi-media Computing (U-Media), 2010 3rd IEEE International Conference on*, pages 198–201, July 2010.

[132] Soonuk Seol Yejin Shin. Smartphone as a remote control proxy in automotive navigation system. *Contemporary Engineering Sciences*, 7(14):683–689, 2014.

[133] Weiqing Zhao, Yafei Sun, and Lijuan Dai. Improving computer basis teaching through mobile communication and cloud computing technology. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 1, pages V1–452–V1–454, Aug 2010.

[134] Huawei: By Xing Zhihao and Zhong Yongfeng. Internet of Things and its future. `http://www.huawei.com/en/about-huawei/publications/communicate/hw-080993.html`, 2010. [Online; accessed February , 2010].