

**A CODE OFFLOADING FRAMEWORK FOR MOBILE CLOUD
COMPUTING: ICEMOBILE**

(MOBİL BULUT BİLİŞİM İÇİN BİR KOD TRANSFER YAZILIM ÇATISI:
ICEMOBILE)

by

Emre ÇALIŞIR, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

Feb 2016

This is to certify that the thesis entitled

**A CODE OFFLOADING FRAMEWORK FOR MOBILE CLOUD
COMPUTING: ICEMOBILE**

prepared by **Emre ÇALIŞIR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering** at the **Galatasaray University** is approved by the

Examining Committee:

Assist. Prof. Gülfem I. ALPTEKİN (Supervisor)
Department of Computer Engineering
Galatasaray University

Assist. Prof. Atay ÖZGÖVDE
Department of Computer Engineering
Galatasaray University

Assoc. Prof. Ali Gökhan YAVUZ
Department of Computer Engineering
Yıldız Technical University

Date: -----

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisors Assist. Prof. Glfem I. ALPTEKİN and Assist. Prof. Atay ÖZGÖVDE for their continuous support during my Master study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors and mentors for my Master study.

Also I thank to my manager Nuri BKREK working in Vodafone Technology for supporting my Master education in the last two years.

Last but not the least, I would like to thank my family: my wife Seda ÇALIŞIR and to my parents for supporting me spiritually throughout writing this thesis and my life in general.

Feb 2016

Emre ÇALIŞIR

TABLE OF CONTENTS

LIST OF SYMBOLS	vi
LIST OF FIGURES	vii
LIST OF TABLES	ix
ABSTRACT	x
RÉSUMÉ	xi
ÖZET	xii
1. INTRODUCTION	1
2. LITERATURE REVIEW	6
3. THE PROPOSED ICEMOBILE ARCHITECTURE	13
3.1 Determining to the Client/Server Communication Model	15
3.1.1 REST Based Computation Offloading	15
3.1.2 Socket Based Computation Offloading	16
3.1.3 RMI Based Computation Offloading	17
3.2 Making the Optimal Offloading Decision	18
3.2.1 Application Analysis: Call Graph Generation	20
3.2.2 Environment and Device Profiling	25
3.2.3 Optimization Model	26
3.3 Modifying the Original Android Application Code	28
4. EXPERIMENTS AND RESULTS	29
4.1 Hardware and Software Specifications of the Implementation Environment	29
4.2 Face Detection Scenario	31

4.2.1 Preparing the Test Images for the Face Detection Scenario	32
4.2.2 Application Analysis of Android-Based Face Detection.....	34
4.2.3 Making the Optimal Offloading Decision in Face Detection Scenario	34
4.2.4 Comparison of REST, Socket and LipeRMI in Face Detection Scenario .	36
4.3 OCR Scenario	38
4.3.1 Preparing the Test Images for the OCR Scenario.....	40
4.3.2 Application Analysis of Android-Based OCR.....	41
4.3.3 Making the Optimal Offloading Decision in OCR Scenario.....	41
4.4 The Scenarios Consisting of Mathematical Calculations	43
4.4.1 Generating a Synthetic Call Graph	45
4.4.2 Results and Evaluation of Optimization Solving in Synthetic Scenarios..	48
4.4.3 Adjusting the Offloadability with Two Practices	50
4.4.3.1 Unconstrained Offloading	50
4.4.3.2 Constrained Offloading	51
5. CONCLUSION	54
6. FUTURE WORK.....	55
REFERENCES.....	56
BIOGRAPHICAL SKETCH.....	58

LIST OF SYMBOLS

API	: Application Programming Interface
APK	: Anroid Application Package
App	: Application
CPU	: Central Processing Unit
HTTP	: Hypertext Transfer Protocol
IP	: Internet Protocol
JAR	: Java Archive
JVM	: Java Virtual Machine
LAN	: Local Area Network
MCC	: Mobile Cloud Computing
OCR	: Optical Character Recognition
OSGI	: Open Services Gateway Initiative
REST	: Representational State Transfer
RMI	: Remote Method Invocation
URI	: Uniform Resource Identifier
VM	: Virtual Machine
WAN	: Wide Area Network
XML	: Extensible Markup Language
WWW	: World Wide Web

LIST OF FIGURES

Figure 1.1: Population vs. Smartphone Sales	2
Figure 1.2: MCC at the Intersection of Three Models.....	3
Figure 2.1: The Cloudlet as Proximate Computing Infrastructure Dedicated to the Mobile Devices	6
Figure 2.2: Dynamic VM Synthesis Process	8
Figure 2.3: MAUI Architecture	9
Figure 2.4: Local and Remote Interfaces.....	10
Figure 2.5: CloneCloud Architecture	11
Figure 3.1: ICEMobile Framework Architecture	14
Figure 3.2: REST Based Communication	16
Figure 3.3: LipeRMI Architecture	18
Figure 3.4: ICEMobile Optimal Decision Making Procedure.....	20
Figure 3.5: A Call Graph Example	21
Figure 3.6: A Call Graph Construction with Soot	24
Figure 4.1: Screenshot of Face Detection Functionality of our Implementation	32
Figure 4.2: Call-Graph Transformation of Face Detection	35
Figure 4.3: Comparison of Energy Consumption of Face Detection Operation	36
Figure 4.4: Comparison of Three Communications based on Time Consumption	37
Figure 4.5: Comparison of Three Communications Based on Energy Consumption ...	37
Figure 4.6: Tesseract OCR Engine Architecture	39
Figure 4.7: Screenshot of OCR Functionality of our Implementation	40

Figure 4.8: Comparison of Time Speed-Up of OCR Operation	42
Figure 4.9: Comparison of Energy Consumption of OCR Operation	43
Figure 4.10: Comparison of Time Speed-Up Server and Client-side Processing	44
Figure 4.11: Call Graph of Matrix Operations	47
Figure 4.12: Call Graph of Unconstrained Offloading.....	51
Figure 4.13: Call Graph of Constrained Offloading.....	52

LIST OF TABLES

Table 1.1: The Advantages and Disadvantages of MCC Using Distant Clouds	4
Table 2.1: Differences of Cloudlet and Cloud	7
Table 2.2: Literature Comparison	12
Table 4.1: Network Bandwidth Comparisons of LAN and WAN.....	31
Table 4.2: Selection of the Images Having Different Dimensions for Face Detection .	33
Table 4.3: Selection of the Images Having Different Dimensions for OCR	41
Table 4.4: Performance Comparison of Matrix Multiplication	45
Table 4.5: Measured Time and Energy Costs of the Call Graph.....	48
Table 4.6: Comparison of Obtained Energy Efficiency of the Scenarios.....	53

ABSTRACT

Today, smartphones have become a crucial part of our life with their high performance data processing features and ability to access information from anywhere at any time. However, with the increasing demand for computation intensive operations on the user side, mobile devices become inadequate to meet the user experience with their limited battery and processing capacities. As a remedy, the concept of mobile cloud computing has been introduced in the literature. Despite the fact that there exist many mobile cloud computing enabled applications in the application markets, the low quality bandwidth and unstable response time of cloud services reduce the user experience. A possible solution may be the use of cloudlets -with the term known in the literature- to offer the cloud services in LAN bandwidth quality by bringing them near the mobile devices. In this thesis, a framework named as ICEMobile (Intensive Computing Environment Mobile) is developed to bring the computation offloading capability into the mobile applications. The application developer can use this framework to determine which methods needs to be offloaded to create an energy efficient mobile execution environment with bounded delays. To prove the advantages of ICEMobile enabled mobile applications in Cloudlet environments, it is implemented both real life and synthetic scenarios with a mobile application and cloudlet software prototype. In this work, we demonstrated that it is possible to make energy savings up to %98 on the mobile device by using the ICEMobile framework.

RÉSUMÉ

Aujourd'hui, les smartphones sont devenus une partie ultime de notre vie avec leurs caractéristiques d'accès à l'information de partout à tout moment et avec haute performance de traitement de données. Cependant, les demandes croissantes d'utilisateurs exigent plusieurs opérations de calcul intensif et les appareils mobiles sont insuffisants pour répondre à l'expérience de l'utilisateur en raison de la capacité de traitement de l'information et des limitations de la batterie. Pour résoudre ce problème, le concept de cloud computing mobile est développé en 2007. Même s'il y a des applications offrant la fonction de cloud mobile dans le marché des applications, la bande passante de faible qualité et le temps de réponse instable de services de cloud computing réduit l'expérience utilisateur. Une solution possible pourrait être l'utilisation de cloudlets -avec le terme connu dans la littérature- d'offrir les services de cloud computing en qualité de LAN à proximité de l'approche aux smartphones. Dans cette thèse, un cadre nommé ICEMobile (Intensive Computing Environment Mobile) est développé pour apporter la capacité de calcul d'offloading dans les applications mobiles. Le développeur de l'application peut utiliser ce cadre afin de déterminer les méthodes qui doivent être déchargés pour créer un environnement d'exécution mobile efficace de l'énergie. Pour afficher les avantages de ICEMobile permis des applications mobiles dans les environnements de cloudlet, on met en œuvre les scénarios de la vie et les scénarios artificiels avec une application mobile et logiciel prototype de cloudlet. Comme expliqué dans cette thèse avec un profond détail, il est possible de créer une efficacité énergétique jusqu'à 98% avec ICEMobile.

ÖZET

Günümüzde akıllı telefonlar bilgiye her an her yerden ulaşabilme ve bilgiyi yüksek performansla işleme özellikleri ile birlikte hayatın vazgeçilmezi hale gelmiştir. Ancak artan kullanıcı deneyimi ile birlikte gelen daha yoğun kaynak kullanımı talepleri ve mobil cihazların kısıtlı bilgi işleme kapasitesi ve bataryası yüzünden kullanıcı deneyimlerini tatmin etmekte yetersiz kalmaktadırlar. Buna çözüm olarak 2007 yılında mobil bulut bilişim teknolojisi kavramı geliştirilmiştir. Uygulama pazarlarında mobil bulut bilişim özelliği olan mobil uygulamalar bulunuyor olsa dahi, düşük bant genişliği ve bulut servislerinin değişken hizmet kalitesi kullanıcı deneyimini olumsuz etkilemektedir. Akademik yazında bulutçuk olarak isimlendirilen ve amacı bulut servislerini mobil cihazın yakınına getirerek yerel ağ bağlantısı kalitesinde hizmet vermek olan yaklaşım bu soruna bir çözüm olarak gösterilebilir. Bu tez çalışmasında, mobil uygulamalara hesaplama transferi özelliği kazandırmayı amaçlayan ICEMobile (Intensive Computing Environment Mobile) isminde bir yazılım iskeleti geliştirilmiştir. Yazılım geliştirici bu altyapıyı kullanarak uygulamanın çalışması enasında enerji verimliliğini sağlamak için bulutçuğa transfer edilmesi gereken fonksiyonları elde edebilir. ICEMobile altyapısının performansını incelemek amacıyla gerçek hayat senaryosu ve yapay senaryo olmak üzere farklı senaryolar gerçekleştirilmiştir. kullanımları bir mobil uygulama ve bulutçuk yazılım prototipiyle gerçekleştirilmiştir. Bu tezde ayrıntılı şekilde anlatıldığı gibi ICEMobile yazılım altyapısı sayesinde %98'e kadar enerji verimliliği sağlamak mümkündür.

1. INTRODUCTION

The evolution of the digital world is correlated with the fulfillment of people's expectations that can be summarized as accessing technology from anywhere and anytime. The technical jargon for this approach is called ubiquitous computing, which is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user (Weiser, 1999). With the increasing mobility of the people and the rise of the social media, people's behaviors are changing towards using small and portable personal computers to benefit from vast resources through Internet. Assuming a fixed increase rate for smartphone sales figure based on 2007-2015 period, we can predict that the number of smartphones sold will surpass the earth population in the first quarter of 2016 and will double the population as of 2017, as shown in Figure 1.1 (Geohive 2015), (Statista, 2015).

The application stores built by major smartphone manufacturers Apple and Samsung opened a new era in mobile communication world: "The Age of Apps". The abbreviated word 'app' is voted as the most popular word of the year by American Dialect Society, in 2010. The applications are basically the computer programs developed with the aim of giving any service to the mobile users, everywhere, anytime. With the latest technology advances, the smartphones are equipped with various sensors, which do not even exist in today's computers. Accelerometer, gyroscope, magnetometer, proximity sensor, light sensor, barometer, thermometer, pedometer, heart rate monitor, fingerprint, radiation detector are some of the examples of the sensors equipped in smartphones. Besides the advances in mobile hardware technology, there are incredible progresses on data science, including big data and machine learning by analyzing petabytes of data, which are unable to be performed by human brain. These advances in hardware and software technology have obvious impacts on all of the industries.

As an example, it is now possible to determine whether a blur on the skin belongs to the skin cancer or not by using a smartphone, in real time.

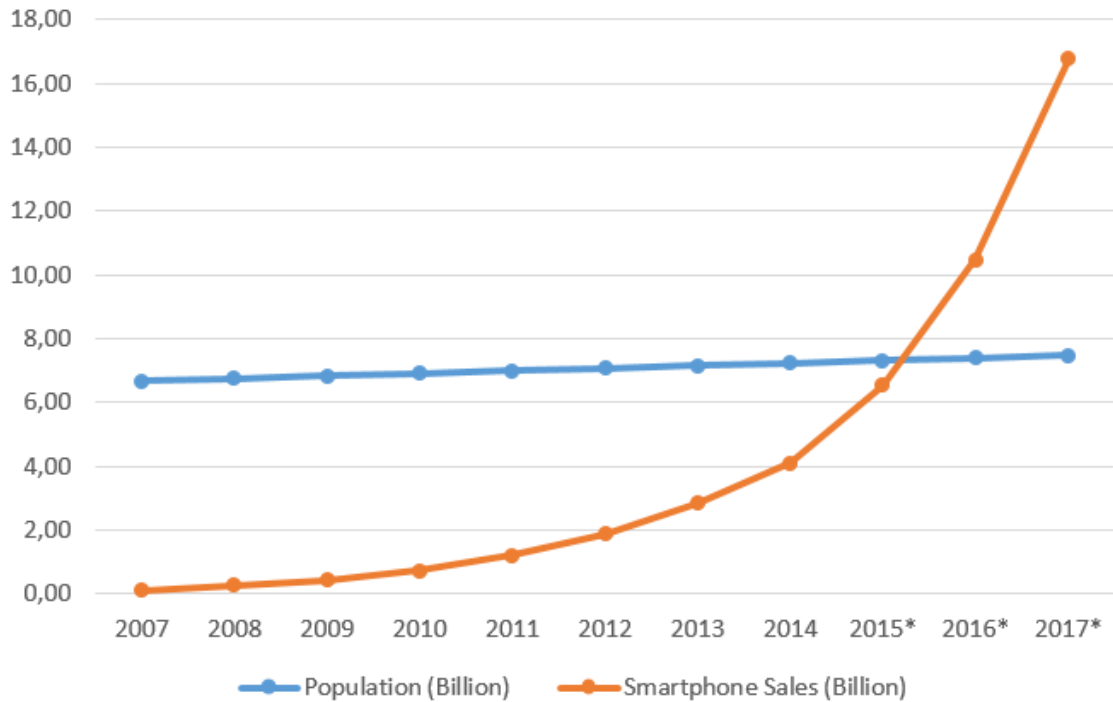


Figure 1.1: Population vs. Smartphone sales ¹

As a result of the growing technologies such as speech recognition, natural language processing, computer vision, machine learning, augmented reality, the need for intense computation is ever increasing. However, current smartphones are unable to respond these needs because of their limited battery and CPU power. Even though the semiconductor technology is constantly evolving, the demand for the computation intensive tasks is higher than the rate of improvement in the mobile hardware technology. To solve this issue, cloud resources can be considered as an ideal candidate.

¹ Each value in the horizontal axis covers the whole year.

A popular definition of Buyya et al., (2009) describe the cloud as a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers. The major cloud providers currently available in the market are Amazon Web Services (AWS), Google, Inc., Microsoft Corporation, Apple, Inc., Salesforce.com, Rackspace Inc., EMC (VMware), IBM, Oracle, and Akamai Technologies, Inc. With the cloud computing, it becomes possible to retrieve on-demand service from a shared pool of configurable computing resources (Mell & Grance, 2011). By combining the three major approaches, namely ubiquitous computing, mobile computing and cloud computing, a new model entitled as Mobile Cloud Computing (MCC) is developed to bring the cloud computing services into the edge to extend resource-rich services on the mobile devices as shown in Figure 1.2 (Abunaser & Alshattnawi, 2011).

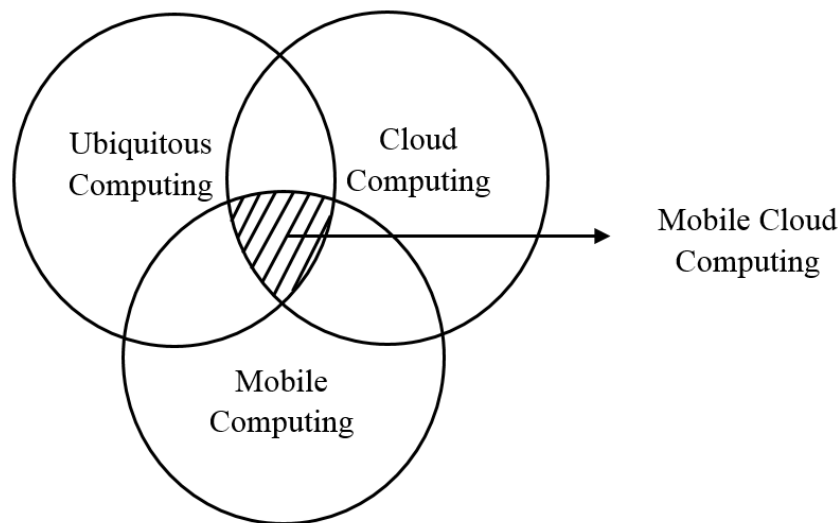


Figure 1.2: MCC at the Intersection of Three Models (Abunaser & Alshattnawi, 2011)

Mobile cloud computing industry is growing very rapidly and is expected to reach over 46.90\$ billion by 2019 (Research and Markets, 2014).

The most used cloud-enabled mobile applications belong to the categories of gaming, entertainment, utilities, education, productivity, business and finance, social networking, healthcare, travel and navigation. The mobile cloud computing industry is continuously evolving to provide better services, however, the requirements of the mobile users can only be partially fulfilled by delegating the execution to the cloud. The main reasons behind this fact is summarized in Table 1.1.

Table 1.1: The Advantages and Disadvantages of MCC Using Distant Clouds

Pros	Cons
Powerful and scalable computers	Unstable bandwidth
Managed by professionals	Unstable quality of service
No initial cost	WAN latencies

A possible solution to the issues in distant cloud based mobile computing is using cloudlets. Cloudlets are small-scale cloud-like infrastructures which are located in one hop distance to the mobile user and connected on a high network bandwidth with and fully dedication to the mobile devices with the aim of executing their computation intensive task (Satyanarayanan et al., 2009). When a mobile device is connected to a cloudlet, it benefits from the much powerful computing capabilities and unlimited energy of the cloudlet. This is also valid for the distant cloud based computation; however, there occurs serious time losses due to the ambiguous service response times of the cloud and the WAN latencies, which negatively affects the user experience in the tasks that require real-time response.

The ways of determining to the convenient computation environment may be:

- When the cloudlet services become unavailable, insufficient or its quality of service is low, the cloud may be a better computing environment.
- When a node contains loops or accesses native APIs, the mobile computing should be preferred rather than offloading.
- When a cloudlet or cloud is available and the network bandwidth is high, the decision may be in the direction of offloading, but it depends on the size of transferred bytes. For larger objects to be transferred, mobile computing may still be a better option than offloading.

In this thesis, we developed a framework by focusing on the well-known code offloading approaches in the literature. The computation or code offloading paradigm means transferring the complex tasks to more powerful environments is used. We developed a prototype having code offloading capability and we addressed many use cases containing the challenges of computation intensive operations. In the background logic of our prototype, there is a decision making process that analyzes the potential gain and loss of the application. To perform this, we analyzed the program (call graph generation) and profiled the amount of time cost of the call graph elements (nodes and edges). This mechanism is tested with an Android application and nearby Java based web server, with the REST web services on the execution of three computation intensive use cases.

2. LITERATURE REVIEW

The studies in the literature that consider the idea of moving the cloud closer have started with the ‘cloudlet’ concept of Satyanarayanan et al. (2009). With the impact of this novel approach, it has become a hot topic in mobile cloud computing related researches for the last years. In the proposed cloudlet model of Satyanarayanan, the cloudlets are at one-hop distant to the mobile device, having high-bandwidth wireless access. They may be located in coffees, airports like wireless hotspots to deliver instant services to the mobile clients from smartphones to wearable devices, as shown in Figure 2.1.



Figure 2.1: The Cloudlet as Proximate Computing Infrastructure Dedicated to the Mobile Devices (Satyanarayanan et al., 2009)

There are several benefits of the cloudlets including stable service quality and high network speed compared to the clouds. Their main differences are shown in Table 2.1 (Satyanarayanan et al., 2009).

Table 2.1: Differences of Cloudlet and Cloud (Satyanarayanan et al., 2009)

	Cloudlet	Cloud
State	Only soft state	Hard and soft state
Management	Self-managed; little to no - professional attention	Professionally administered, 24x7 operator
Environment	“Datacenter in a box” at business premises	Machine room with power conditioning and cooling
Ownership	Decentralized ownership by local business	Centralized ownership by Amazon, Yahoo!, etc.
Network	LAN latency / bandwidth	Internet latency / bandwidth
Sharing	Few users at a time	100s – 1000s of users at a time

The proposed computation offloading techniques of Satyanarayanan et al. (2009) are VM migration and dynamic VM synthesis. In VM migration, the entire snapshot of the mobile device is transferred to the cloudlet, while in the second technique, there is a dynamic VM synthesis to reduce the size of the VM snapshot by receiving the delta with the previous VM state (Figure 2.2). In this procedure, all of the mobile operations are performed at the cloudlet after the initial synchronization of mobile device and cloudlet.

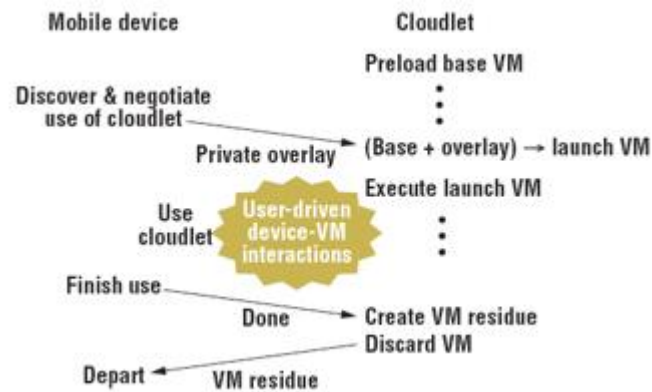


Figure 2.2: Dynamic VM Synthesis Process (Satyanarayanan et al. 2009)

The dynamic VM synthesis technique provides offloading capability for each mobile device since it transfers the VM of the mobile device rather than platform specific objects. On the other hand, it contains many problems in the following areas:

- The user is blocked until the operation is completed.
- There is not any decision making procedure on offloading; it always transfers the VM.
- The VM overlay does not only contain the computation intensive code blocks, but also lots of unnecessary data that present the VM state.

Another research entitled as MAUI targets to offload only the computation intensive parts of the mobile application with method level offloading based on the .NET framework (Cuervo et al., 2010). In this research, a system is proposed which is capable of making the decision of whether determining whether offloading will save energy for each offloadable part of the application. The proposed method-level offloading type is a more fine-grained approach than the VM migration or dynamic VM synthesis. In addition to reducing size of the transferred objects, MAUI also provides an intelligent mechanism targeting to reduce the energy consumption on the mobile device by optimally deciding to offload subject to device and server capabilities and network conditions (Figure 2.3). In this architecture, there is a synchronized runtime environment in both client and mobile device, and necessary offloading operations are executed via remote procedure calls.

In order to create code offloading functionality, there are some manual processes that reduce the applicability and usability of the MAUI framework. The first issue is having the necessity of modification on the application since it seems as a disadvantage comparing with (Satyanarayanan et al., 2009). Secondly, there is not any mechanism for automatically identifying the computation intensive parts of the application.

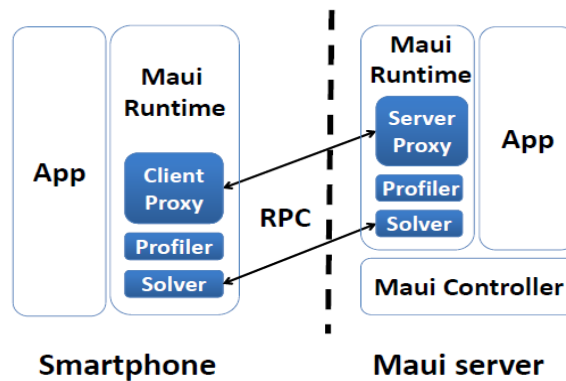


Figure 2.3: MAUI Architecture (Cuervo et al., 2010)

In MAUI approach, there is a need for program partitioning since it has the capability of fine-grained offloading. In the programming environment of MAUI, the application developers insert the *@remotable* annotation in the beginning of all methods, which they find appropriate for offloading. These methods should not contain any calls to the mobile device specific interfaces such as sensor APIs and the form elements of the mobile application. The component named as MAUI Solver takes into account only the annotated methods. MAUI is designed to support the applications written for the Microsoft .NET Common Language Runtime, which is the virtual machine component of the .NET framework that manages the execution of the programs. The advantage of using CLR is that it enables execution on various smartphones having different instruction set architectures, on desktops and on servers. MAUI extracts the remotable methods by using reflection with the custom attribute feature of .NET Reflection API, which automatically identifies the *@remotable* annotations.

At compilation time, MAUI generates a wrapper for all remoteable methods by adding additional parameters to save the return values. (Figure 2.4)

```

//original interface
public interface IEnemy {
  [Remoteable] bool SelectEnemy(int x, int y);
  [Remoteable] void ShowHistory();
  void UpdateGUI();
}

//remote service interface
public interface IEnemyService {
  MAUIMessage<AppState, bool> SelectEnemy (AppState state, int x, int y);
  MAUIMessage<AppState, MauiVoid> ShowHistory(AppState state);
}

```




Figure 2.4: Local and Remote Interfaces (Cuervo et al., 2010)

Another mobile cloud computing research entitled as ThinkAir provides with a framework based on method level offloading with the virtual machine technology (Kosta et al., 2012). With its novel architecture, it solves the non-scalability problems of MAUI. ThinkAir enables the cloud features of on-demand resource allocation and parallelism in mobile cloud computing. However, to implement the ThinkAir for execution offloading, the developer still needs to modify the existing application code, which is the same case in MAUI.

Another research entitled as CloneCloud focuses on the application partitioning and thread level offloading (Chun & Maniatis., 2009). The VM migration mechanism is used in the background of this study by offloading execution blocks of applications from smartphones to their mirror image running on the server. This framework works as a middleware in Android operating system, on the top of Dalvik VM.

An advantage of the CloneCloud is that it does not require code modification on the application instead, it automatically detects the right offloadable portions of the code. However, the executable application should be recreated with the CloneCloud. In this rewriting process, the migration points are placed into the mobile application code as an output of the static analyzer.

In the application runtime, if the application flow reaches to migration point, the individual threads migrate from the mobile device to the clone VM. The mobile user is not blocked until the migrated thread returns back and merge with the original state since the remaining functionality of application flow keeps running. The static analysis and dynamic profiling operations are operated on the partition analyzer component, which is running on the nearby cloud (Figure 2.5). In this architecture, the application VM is existing in both server and mobile device to create an ideal environment for offloading.

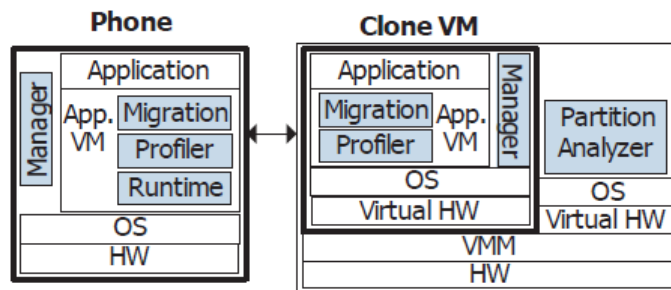


Figure 2.5: CloneCloud Architecture (Chun & Maniatis., 2009)

Based on the CloneCloud, Zhang et al. (2010) have introduced an elastic application programming framework having component-level offloading capability. The components are named as weblets, that are autonomous software entities running either on the device or cloud and exposing RESTful web service interfaces via HTTP. In this study, the decision of where to launch a weblet is given according to the decision given by Naive Bayesian learning technique. Some of the features used in this machine learning algorithm are the device CPU, battery level, application configuration and the results of the previous offloads.

The second research that is based on component-level offloading is OSGI based offloading (Verbelen et al. 2012). OSGI is a service oriented application partitioning management system allowing dynamically loading and unloading software modules. The target of this study is to create dynamic cloudlet networks.

In this architecture, all devices in the LAN network can cooperate with each other by assigning the roles of node agent or cloudlet agent. In a local network, the node with the more resource is assigned the cloudlet agent role. If the cloudlet agent is connected to the Internet, it can connect with the cloudlet agents of different cloudlets.

In addition to the previous studies, a research, named as Cuckoo, focuses creating a dynamic offloading decision making tool in runtime in Android OS installed smartphones (Kemp et al. 2010). It presents a system to offload mobile device applications onto a cloud using a Java stub/proxy model. Cuckoo can be offloaded onto any resource that runs the Java Virtual Machine. To use Cuckoo, the applications need to be re-written such that the application supports remote execution as well as local execution. Moreover, it does not contain any decision making to offloading in this research, it always offloads when it connects to the cloud.

Regarding the researches stated above, Table 2.2 shows their strengths and weaknesses in mobile cloud computing architecture. CloneCloud seems the ideal offloading platform among all frameworks with its thread-level granularity and applicability.

Table 2.2: Literature Comparison

	Programmer Effort	Granularity of Offloading	Target Mobile Device	Makes Offloading Decision
Dynamic VM Synthesis	No	VM-level	All	No
CloneCloud	No	Thread-level	Android	No
MAUI	Yes	Method-level	All	Yes
Cuckoo	Yes	Method-level	Android	No
Weblet	Yes	Component-level	All	Yes
ThinkAir	Yes	Method-level	Android	Yes
OSGI	Yes	Component-level	All	Yes

3. THE PROPOSED ICEMOBILE ARCHITECTURE

The aim of our proposed framework ICEMobile is to minimize the energy consumption of the mobile device when computation intensive functions needs to be executed. To realize this purpose, it transfers the resource intensive code partitions of mobile application to the cloudlet for their remote execution. The main purpose is to extend battery life of the mobile device; hence, the energy consumption of the cloudlet is not the concern, since it is continuously fed from the energy sources.

The ICEMobile framework architecture is depicted in Figure 3.1. It involves RMI framework situated both in the mobile platform and in the cloudlet. In case of cloud-based offloading, it is possible to integrate this framework into the cloud configuration. The advantage of using the nearby cloud in LAN provides with higher bandwidth compared to a one in WAN. Besides, it does not require having Internet connection since the cloudlet is ready to satisfy all client needs. If more resources are needed, the distant cloud services may be used, but the user experience will surely be decreased when real-time computation intensive operations are executed.

The essential part of the ICEMobile offloading environment is in the server-side, where there is not any limitation on the operating system with the help of JVM technology. The environmental profiling and program analyzing efforts together with the optimization component in the server side enable optimal decision making for offloading process (The decision making model will be described in detail in Section 3.2.) In our architecture, the Profiler and the Analyzer are not executable programs. They are manually operated by the programmer and their results are transmitted to the optimization solver as an input. In the literature, there exist many examples that makes them automatically (Chun & Maniatis., 2009). In addition to these components, the same offloadable methods of the mobile application are also present on the cloudlet to be executed when necessary.

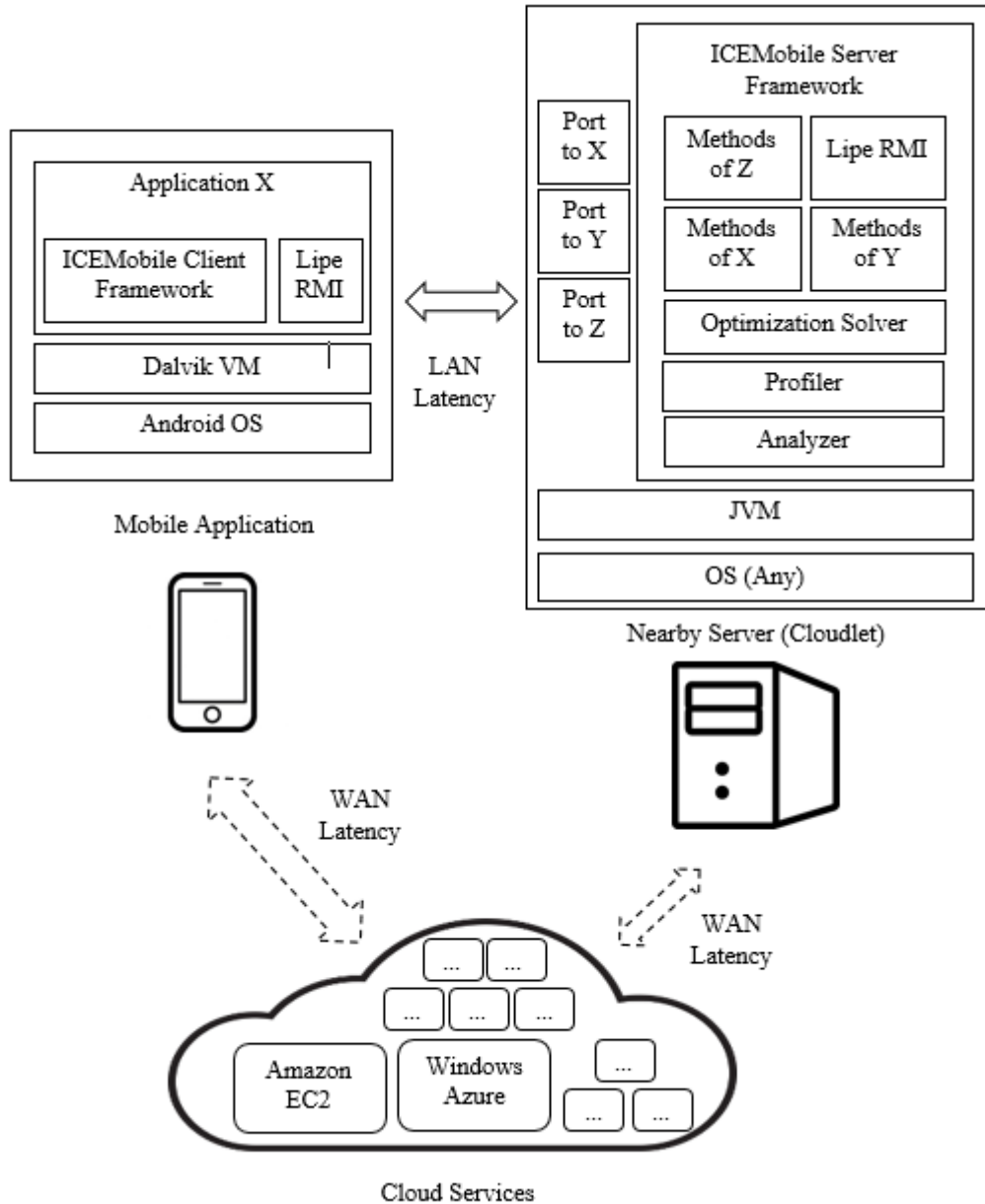


Figure 3.1: ICEMobile Framework Architecture

In the client side, any mobile device can benefit from the ICEMobile task offloading mechanism. For mobile devices with higher computational capabilities the need for offloading tends to decrease. In order to promote offloading, the application code needs to be modified with the ICEMobile Client framework codes as explained in detail in Section 3.3.

In the server side, a port is specifically reserved for each client. In order to initiate the server, the following two steps should be proceeded:

- a. A *CallHandler* is identified. The interface file that keeps the method signatures and the class file that contains the client methods are registered globally to the *CallHandler*.
- b. The *CallHandler* is bounded with an available port to start listening the environment and respond the incoming requests.

3.1 Determining the Client/Server Communication Model

The most commonly used communication types in the classic MCC architectures are the REST-based communication, HTTP-socket based communication and RMI-based communications. All these techniques are applied on the scenarios describe in Section 4 to determine the most efficient one.

3.1.1 REST-based Computation Offloading

REST architecture-based web services are the most frequently used communication services of the WWW. They are used with the HTTP verbs of GET, POST, PUT, etc. Their interfaces are identified by URI to ensure uniqueness. RESTful web services are also one of the essential communication types in client/server-based mobile cloud computing operations. As described in the research of Christensen (2009), RESTful web services provide a straightforward communication to the smart mobile device environments. They are easy to invoke, produce a discretely formatted response (no dangling TCP connection). Figure 3.2 shows the REST-based communication in a traditional client/server architecture.² The HTTP packet contains the information about the type of the request and the data itself.

² Stateless Web Service Conversion via REST. http://clean-clouds.com/?attachment_id=98

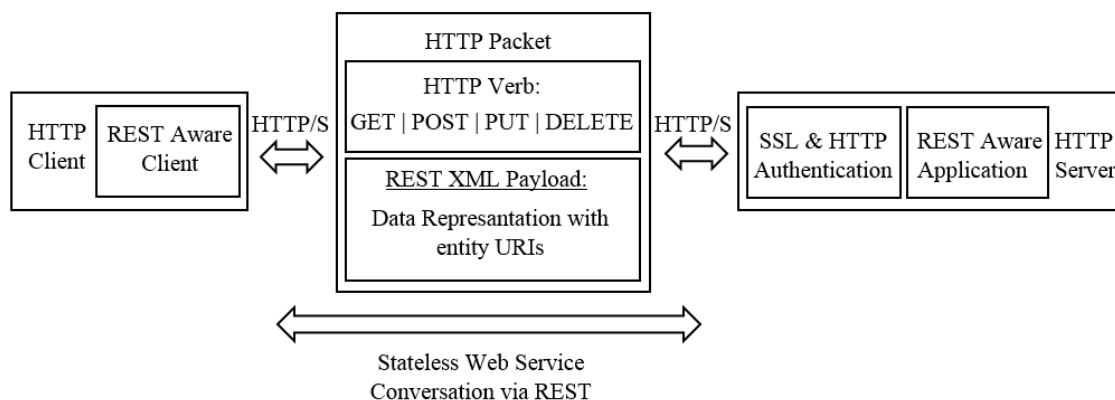


Figure 3.2: REST-Based Communication

The key aspects of REST that make it attractive for mobile applications are as follows:

- REST is stateless,
- REST is URL-based,
- REST responses are usually HTTP-based,
- REST delivery can be made very succinct.

In order to analyze the impact of the REST services in mobile cloud computing, we implemented Jersey RESTful web services for the face detection and OCR scenarios.³ In our implementation, we used Apache Tomcat, which is an open source web server that was developed by Apache Software Foundation. Apache Tomcat provides with a pure Java HTTP server environment for Java code to run in.⁴ The performance of the RESTful web services on these scenarios are explained in Section 4.

3.1.2 Socket-Based Computation Offloading

A socket is a software endpoint that establishes bidirectional communication between a server program and one or more client programs.⁵ In this technique, a server program provides a specific port to respond incoming requests from the client programs.

³ Jersey RESTful Web Services in Java. <https://jersey.java.net/>

⁴ Apache Tomcat. <http://tomcat.apache.org>

⁵ Socket Communications, Oracle. <http://www.oracle.com/technetwork/java/socket-140484.html>

The `java.net` package provides two classes called *Socket* and *ServerSocket*, which represent the client side of the connection and the server side of the connection, respectively. The obtained results are examined in detail in Section 4 in detail.

3.1.3 RMI-Based Computation Offloading

The working principle of code offloading in ICEMobile framework is based on the client/server architecture with remote method invocations, which is a simple and directed model to distribute Java objects. Since Android software development platform does not support the RMI part of Java, we used an open source third party RMI framework called LipeRMI.⁶ This framework is totally independent from native Java RMI. One of the key feature of the LipeRMI framework is that it uses an Internet optimized approach for communication layer with only one port per client as shown in server binding operation (Figure 3.3).

Dedicating a specific port to a unique mobile client may be an advantage on the perspective of computation intensive applications such as voice/face recognition, gaming, etc. This approach is similar to the architectures of Satyanarayanan et al.(2009), Chun & Maniatis (2009), Cuervo et al. (2010): When ICEMobile framework enables client and server locating in one hop distance to each other and communicating over a specific HTTP port, it forms the essential value of the cloudlet-based offloading of a dedicated service providing. The main difference is that the connection initiated by the client remains active with the server throughout the session. Moreover, LipeRMI compresses all of the transferred data, which decreases the bandwidth requirement of Java RMI.

In the upper side of the architecture, the abstraction of LipeRMI enables using objects in both server and client side, with the help of CallHandler. All transferrable objects and their remote implementations are identified at the CallHandler object. Any application that uses LipeRMI can act as client or can open its objects to other clients.

⁶ LipeRMI <http://lipermi.sourceforge.net/>

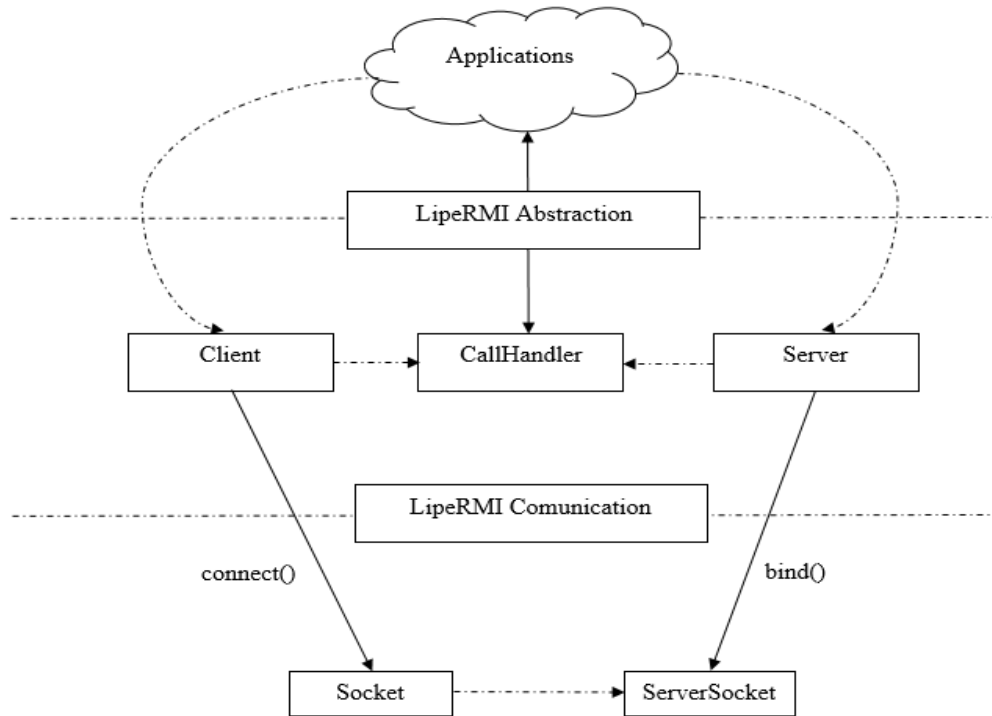


Figure 3.3: LipeRMI Architecture

3.2 Making the Optimal Offloading Decision

The ICEMobile framework gives the offloading decision based on the perspective of providing energy efficiency on the mobile device. The first parameter that affects the offloading decision is related with the environmental conditions, which is time and energy cost of each call graph element. In addition to this parameter, the application algorithm has an impact on offloading. Since our optimization function targets an overall energy efficiency, the flow of call graph directly affects the offloading decision. A node that contains a lightweight process may be offloaded due to the fact that the cost of its consecutive nodes are heavier.

The optimization solver in the ICEMobile framework aims at determining which portions of the application code are better to be offloaded to the remote server in order to use resources more effectively with a similar mechanism of Cuervo et al. (2010). Realizing the offloading in method-level granularity provides transferring only the computation intensive parts of the application, rather than transferring the full VM snapshot like the Kimberley architecture (Satyanarayanan et al., 2009). As mentioned in the previous section there are many operations to be completed to make the offloading decision. The ICEMobile optimization process is shown in Figure 3.4, which can be summarized as the consecutive steps of application analysis, environmental and device profiling, and optimization solving.

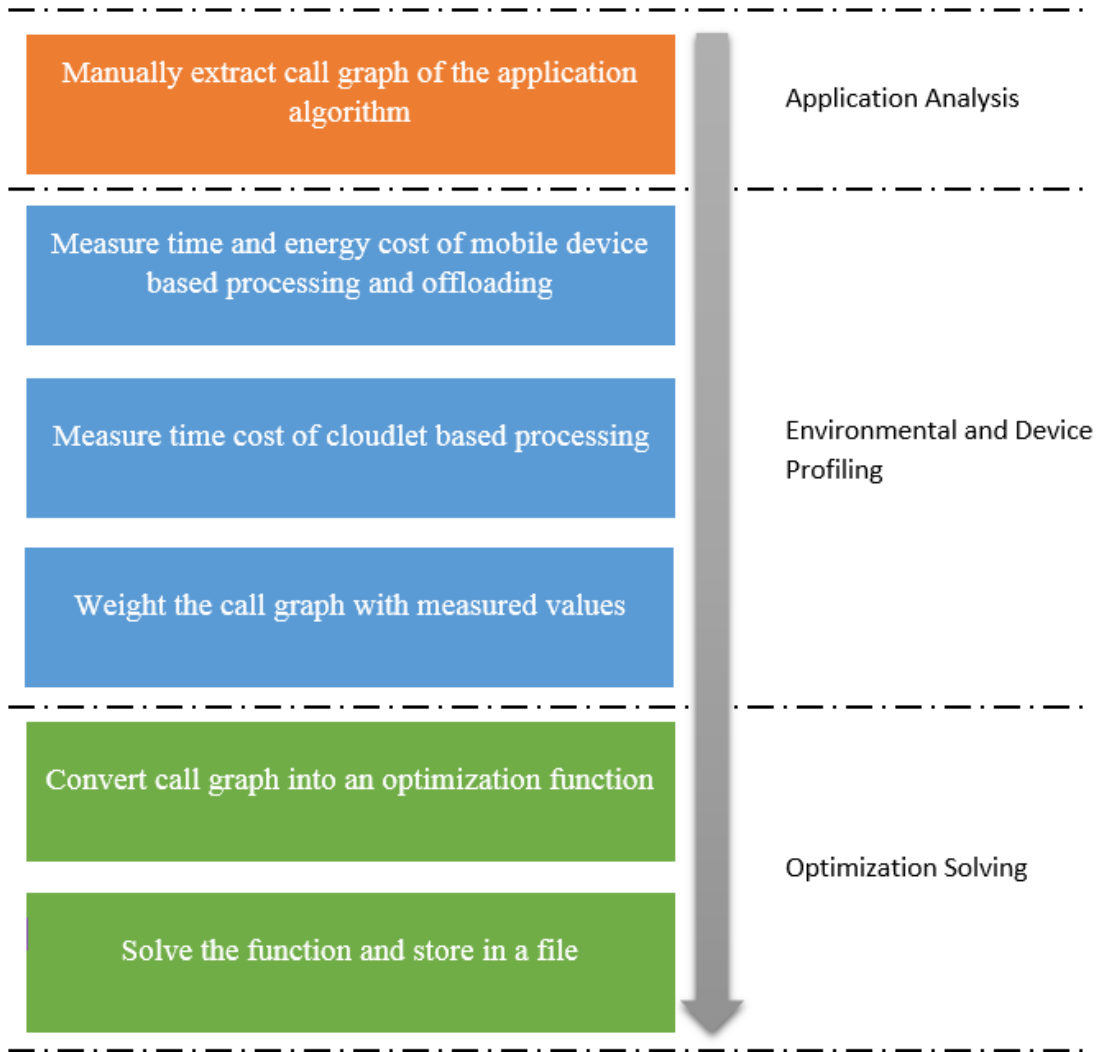


Figure 3.4: ICEMobile Optimal Decision Making Procedure

3.2.1 Application Analysis: Call Graph Generation

The initial step of the decision making process is analyzing the application code to extract related call graph. Call graph, i.e. control flow graph or call chain graph, contains the hierarchies in application algorithm. For the sake of simplicity, the term *call graph* will be used throughout this study. The advantage of extracting a call graph is that it simplifies analyzing the weights of the application classes, the points where the energy consumption or latency in time are maximum.

Figure 3.5 shows an example of a call graph that is composed of nodes and edges.⁷ In this example, there are 7 nodes and 8 edges with their weights defined with time cost and number of method calling.

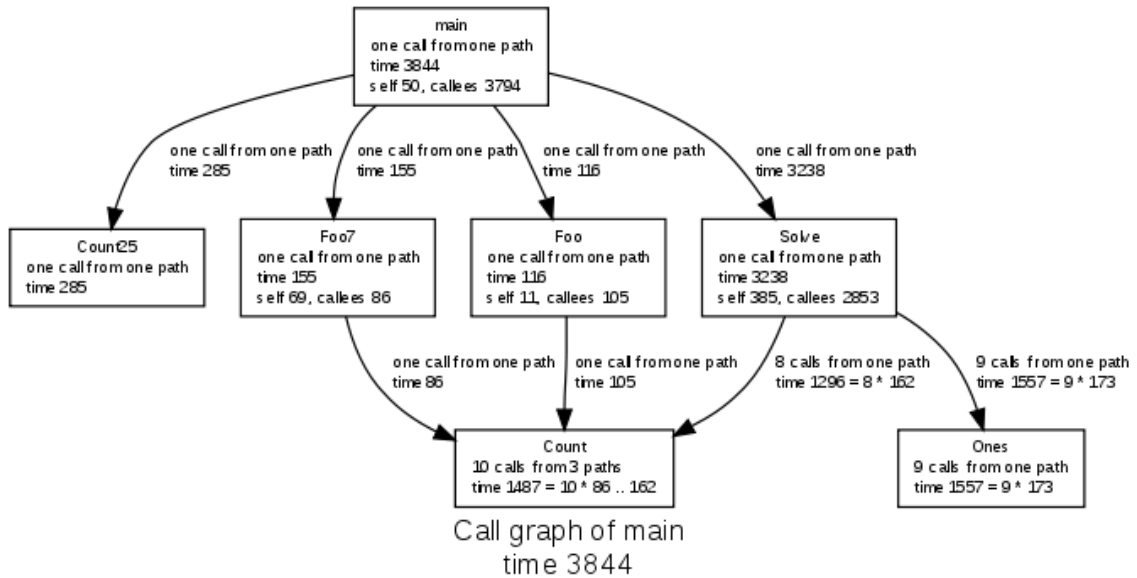


Figure 3.5: A Call Graph Example

In the call graph, the role of the nodes is to present its associated method names and their weights in the term of time and energy cost. On the other hand, the edges describe the calling relationship between the classes with the time cost and their associated weights. Those weights describe the time and energy cost of offloading. Alternatively, there may be another weight descriptions such as number of methods invocations, loops.

Although there are various application analysis tools, their working principle can be classified in two groups: Static and dynamic call graph analysis (Rountev et al., 2004). Static generators extract all of the hierarchies of the application software. They read the classes from a JAR or APK file and walk down the method bodies to generate a file containing calling relationships.

⁷ Call Graph. <http://www.bound-t.com/targets/avr/example-brochure/call-graph.html>

The most commonly used static call graph generator is Soot, which is an optimization and a language manipulation framework for the Java programming language.⁸ In this thesis, many trials with the Soot framework are realized to construct the call graph of the mobile application. However, the disadvantage of using such a framework is that it goes into too much detail, since it continues to partition the application until the lowest sub-segments, such as creation of the variables...etc. are found. Figure 3.6 shows an example of ICEMobile call graph that is constructed with the Soot framework. This example proves the inefficiency of building graphs having too much elements including definition of the variables, value assignments...etc. Another call graph constructor framework, called Java-CallGraph produces static and dynamic call graphs for Java programs.⁹ The difference of dynamic call graph generation is that it runs a Java agent during application runtime and instruments the methods of a user-defined set of classes in order to track their invocations. At the termination of JVM, the framework generates the collected results including number of calls information. This tool gives too much details as the output, such as Soot framework.

In addition to the problems of complexity stated above, it is very important to avoid from partitioning some parts of the application because of their transfer constraints. For instance, the methods accessing to the native functions of the mobile device such as capturing an image using the mobile device camera or receiving an input from the user, are not suitable for offloading. Another constraint is on the methods that contain wrappers to other native languages, such as the functions containing the OCR features of Tesseract library that is written in C language.¹⁰ They should not be partitioned. In order to overcome these issues, the call graph is generated manually, as described in Section 4.

After generating the call graph, the profiler component measures the time spent for each node and edge in the call graph. The merge operation of the call graph objects with their associated weights allow the creation of the optimization model. The role of the optimization solver component is the execution of the optimization model that includes the objective function and related constraints.

⁸ Soot <http://sable.github.io/soot/>

⁹ Java Call-Graph <https://github.com/gousiosg/java-callgraph>

¹⁰ Tesseract <https://code.google.com/p/tesseract-ocr/>

The model may be built as a maximizing problem of the battery life, or as a minimizing problem of the energy consumption or latency problem. The optimization problem is written in R-language, by using the input parameters received from the profiler. The optimization problem will be presented in detail in Section 3.2.3.



Figure 3.6: Call Graph Construction with Soot

3.2.2 Environment and Device Profiling

The context-awareness is a key feature of call graph-based optimization frameworks to make the offloading decision. The context awareness means measuring various parameters in the current environment such as bandwidth value, number of users connected to the cloudlet, current cloudlet service performance, performance of the previous offloads, processing capability of the mobile device, battery of the mobile device, etc. In the call graph-based optimization architectures, these parameters have crucial impact on the offloading decision making. In order to show the performance of the introduced infrastructure in terms of CPU speeds of the mobile device and the cloudlet, it is chosen to benefit from the Linpack benchmarks, which is a measurement of a system's floating point computing power.¹¹ It evaluates how fast a computer solves a dense n by n system of linear equations that is a typical task in engineering.

On the other hand, measuring the bandwidth value may not be enough to determine network quality, even though it is very easy to get the bandwidth value with the Android WifiManager APIs.¹² To determine the connection speed in a network, the best practice seems as sending a trial packet with the known size and calculating the difference between the sending and receiving timestamps. The round trip time can be accurately calculated by dividing the calculated value to the packet size. These are the prominent approaches of the environmental profiling in mobile cloud computing research. However, the profiling procedure in ICEMobile architecture contains even simpler calculation. In this process, the role of the programmer is to manually measure how much time is spent during the execution of each node and edge, by performing each scenario on the cloudlet and the mobile device. This reveals more compact information than Linpack benchmarks and the packet round trip times.

¹¹ Linpack Benchmark <http://www.netlib.org/benchmark/linpackjava/>

¹² Android Wi-fi Manager <http://developer.android.com/reference/android/net/wifi/WifiManager.html>

The generated measurements are given in Section 4. Related calculations for the nodes and edges are as follows:

- Perform all methods on the device and measure the time and energy cost
- Perform all offloadable methods on the cloudlet and measure the time cost
- Perform offloading for all offloadable methods and measure the time and energy cost

A Java code that retrieves the actual timestamp in milliseconds is placed at the beginning and the end of the methods in each part to measure the time cost. Since the values vary in different trials, the operations are executed ten times with the help of loops and the average time cost is calculated automatically at the end of each loop.

The energy cost is measured using the Android application called PowerTutor.¹³ It is an energy profiler for Android developed at University of Michigan, with the support of Google. The most prominent feature of PowerTutor is that it can measure the actual energy on the hardware and application basis. It is the one of the most efficient application to measure energy consumptions (Bakker, 2014). In the implementation part of this thesis, both the energy consumption of Wi-Fi component when offloading the nodes to the cloudlet and the energy consumption of CPU when processing the nodes on the mobile device are measured.

At the end of the measurement step and determination of the parameter values, the values are attached to nodes and edges of the graph. This call graph is the input of the optimization function.

3.2.3 Optimization Model

The next step is converting the call graph to be suitable for 0-1 integer linear programming problem. In this model, it is assumed that the nodes are the function variables and the weights are their coefficients.

¹³ PowerTutor <http://ziyang.eecs.umich.edu/projects/powerTutor/>

At the end of the optimization process, the nodes that are better to be offloaded will be determined. The first step of the optimization is to determine the objective function.

The offloading variables set for each node can take two values: 0 for not offloading and 1 for offloading. In the ICEMobile architecture, the optimization model is using the *lpsolve* solver of the R language.¹⁴

For a given graph $G = (N, E)$, N represents node and E represents edge. Each node $n \in N$ represents a method and edge $e = (m, n)$ represents an invocation of method n from m . We annotated each node $n \in N$ with the energy it takes to execute the method locally E_n^l . The energy consumption of the cloudlet is not considered. The time a node takes to execute the method locally is shown by T_n^l and the time a node takes to execute the method remotely is specified by T_n^r . Each edge is annotated as $e = (m, n)$. The time it takes to transfer the necessary program state is given by $B_{m,n}$ when m calls n and the energy cost of transferring that state is annotated as $C_{m,n}$. The binary parameter r_n indicates whether the node n is offloadable or not.

The 0-1 integer programming function of ICEMobile is shown in the equations (1), (2) and (3). The objective is to maximize the energy savings in the mobile device (1). The optimization solver determines the offloading variables (I_n) that is the indicator variable of offloading decision for each node. As a result of the optimization, if I_n is equal to 0, it means that the method will not be offloaded and if I_n is equal to 1, the method will be offloaded and it will result in saving energy.

¹⁴ R *lpsolve* <https://cran.r-project.org/web/packages/lpSolve/index.html>

$$\max \sum_{n \in N} I_n x E_n^l - \sum_{(m,n) \in E} |I_m - I_n| x C_{m,n} \quad (1)$$

$$s. t. \sum_{n \in N} ((1 - I_n) x T_n^l) + (I_n x T_n^r) + \sum_{(m,n) \in E} (|I_m - I_n| x B_{m,n}) \leq L \quad (2)$$

$$and I_n \leq r_n, \forall n \in N \quad (3)$$

In the first term in the objective function (1), it is clear that the energy cost of local executions (executions on the mobile device) is taken into account only when the offloading decision is true. Consequently, it shows the amount of total energy saved by executing the methods remotely. The second term in the objective function illustrates the energy cost of data transfer required for executing a method remotely. The important point is that when two consecutive nodes are executed on the same instance (on the mobile device or on the cloudlet), the energy cost of data transfer will be equal to zero. The first constraint (2) defines that the total time to execute the program should be within a specified interval, L . It is the sum of the local and remote time consumptions and the amount of time needed when two consecutive nodes are executed in different places. Finally, the second constraint limits the possible values of the offloading variable (3).

3.3 Modifying the Original Android Application Code

The ICEMobile client framework contains the RMI interfaces presenting the signatures of the offloadable methods and the necessary codes that creates a connection with the cloudlet. It requires adding the offloading logic to the beginning of the offloadable methods. The ICEMobileDecisionMap object contains the offloading decision of each offloadable method. It is placed into the mobile device memory by reading from the optimization result text file and putting to a Java Hash Map object having $\langle key, value \rangle$ pairs.

4. EXPERIMENTS AND RESULTS

At this section, we present and then analyze the offloading decisions of different applications in different domains by using our proposed ICEMobile framework. We developed the following resource intensive use cases: Face detection, OCR and mathematical calculations. In order to give the offloading decision in each scenario, we made the measurements of time and energy and we created related call graphs. We built the optimization function using obtained values and we solved the 0-1 integer linear programming in R. Finally, we analyzed the results upon the offloading decisions.

4.1 Hardware and Software Specifications of the Implementation Environment

The backend server is a notebook computer having a strong computation capability with its 4 cores of Intel i7 fourth generation x64 microprocessor and a RAM with the 8GB DDR3 capacity. The operating system is Windows 7. Apache Tomcat provides the web server functionality by listening incoming requests.

As the mobile client device, we used an LG G3 smartphone having Android Kitkat 4.4.2 operating system, a Qualcomm Snapdragon 801 microprocessor with quad core processors hardware and a 3GB RAM.

In order to realize the client/server communication, we included the Jersey library for REST-based offloading, native Java socket APIs for socket based offloading and Lipe-RMI for RMI-based offloading. The offloading details for each communication model are explained in detail in the following use case sections.

In the client side, we developed an Android application that contains the operations of our uses cases.

We used native FFTE Android API for face detection use case, tess-two library for OCR use case and JScience library for mathematical calculations use case.^{15,16,17} In the server side, we developed a backend server software in Java which contains the same methods of the Android application with similar libraries: OpenCV for face detection, tess4j for OCR and JScience for mathematical calculations.

In the energy measurement part, we used the most up-to-date version of PowerTutor, named as PowerTutor 2 Pro, which is a mobile application compatible with our Android device.

In addition to the hardware and software requirements of the cloudlet environment, the network quality is also an important criterion that affects the offloading performance. In our implementation, the mobile device and the server are connected to the same LAN to achieve a better network speed compared to WAN. In order to make a comparison between two networks, we developed a function that measures the bandwidth of the LAN and WAN connections. There are various methods of measuring the network bandwidth. The most known technique is to transfer a small packet and calculate the amount of time spent until its arrival. In our test, we preferred to transfer a zip file having the size of 50 MB to avoid temporary changes in the bandwidth. The background flow of the “*Check bandwidth*” functionality initially measures WAN bandwidth by establishing a connection with a known web address on which there is a downloadable zip file. In order to calculate the WAN bandwidth, the number of total downloaded sequence byte is divided to the amount of the time passed. In the second step, it calculates the LAN bandwidth value with the similar process. At this step, the mobile client downloads the same zip file from the cloudlet web server. In conclusion, it finally obtains the accurate values of WAN and LAN bandwidths as shown in Table 4.1. As a result of our tests, the calculated LAN bandwidth is nearly %64 higher than the WAN bandwidth, which proves the benefits of using nearby clouds to realize more effective communications.

¹⁵ OpenCV <http://opencv.org/>

¹⁶ Tesseract <https://code.google.com/p/tesseract-ocr/>

¹⁷ JScience <http://jscience.org/>

Table 4.1: Network Bandwidth Comparisons of LAN and WAN

Network Type	Megabits per second
LAN	11.94
WAN	7.25

4.2 Face Detection Scenario

This is one of the most commonly applied scenarios for the mobile cloud computing. It is basically the identification of human faces in a given photo. Even though it seems as a very simple image processing operation, with the advances of camera technologies and its reflections on the smartphones, the face detection operation becomes a highly computation intensive operation when the captured high resolution image is not resized.

There are several image processing libraries that contain face detection functionalities in the computer vision. In the Android part, we used the native Android FFTE face detection API. Our reason for choosing FFTE rather than OpenCV Android SDK is that the APIs of OpenCV are not detecting faces of a static image. Instead, they make the face detection when there is an actual streaming on the camera.¹⁸ This was a contradictory functionality with the cloudlet software, in which we used the classical OpenCV framework, which is a robust image processing library. It is designed for computational efficiency with a strong focus on real-time applications and written in optimized C/C++ and can benefit from the multi-core processing capabilities of the CPU. Figure 4.1 shows the screenshot of our Android application implementation. The user selects a photo from the hard drive and then selects face detection function. The operation is performed on mobile device or on synchronized cloudlet.

¹⁸ OpenCV4Android SDK, <http://opencv.org/platforms/android.html>

Then the faces are drawn with green rectangles based on the obtained coordinate values of the detected faces. In the right side of the mobile application, the time cost is shown with the detail of the call graph elements.

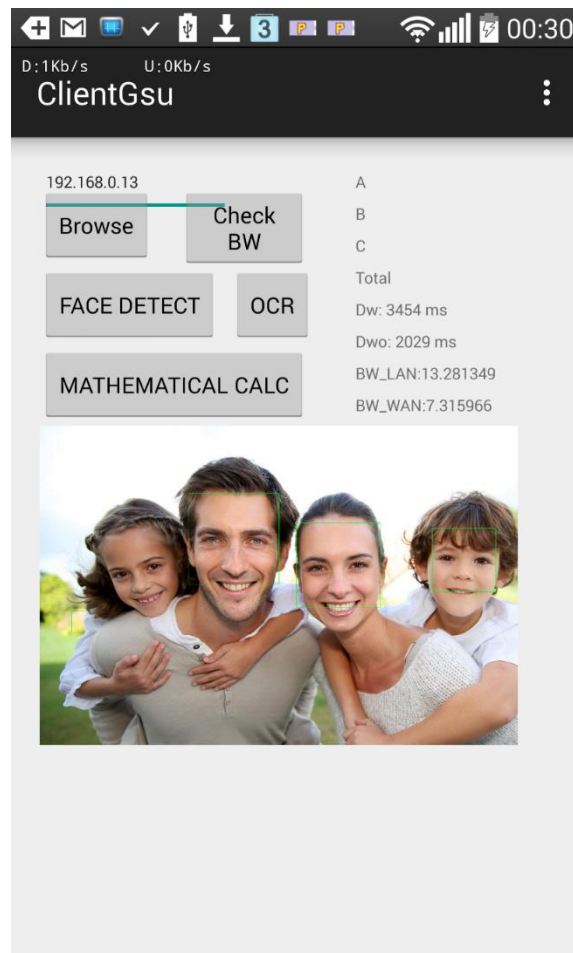


Figure 4.1: Screenshot of Face Detection Functionality of our Implementation

4.2.1 Preparing the Test Images for the Face Detection Scenario

In order to make accurate comparisons about the computation performance of the mobile device, we chose an image containing human faces and we resized this image to obtain multiple images in different size of pixels.

The main reason of duplicating an image in different sizes is to eliminate the other parameters such as RGB values of the pixels, which could affect the test performance. Table 4.2 shows the general properties of the images. Here, the total number of pixels describes the multiplication of the width and height values.

Table 4.2: Selection of the Images Having Different Dimensions for Face Detection

	Width	Height	Total Number of Pixels	Number of Faces	Size on Disk
Image1	720	480	0.3M	4	102KB
Image2	1440	960	1.3M	4	321KB
Image3	2880	1920	5.5M	4	980KB
Image4	5760	3840	22.1M	4	3410MB

4.2.2 Application Analysis of Android-Based Face Detection

The face detection application is composed of the steps below as shown in the On-Device Processing section of Figure 4.2.

- a. **Image Selection:** The image is selected from the gallery or captured using camera. This is not an offloadable part of the application since it needs to use Android view components.
- b. **Preparation of Image Properties for FFTE:** The input image is converted into a bitmap object and the necessary data preparation for FFTE function is done.
- c. **Face Detection Operation:** FFTE API is called with the given bitmap and the information containing the number of faces and their coordinates is received.
- d. **Displaying the Results:** After getting the information containing the number of detected faces and their exact locations in (x, y) coordinates, the rectangles drawn based on the calculated coordinates. This operation is related with the Android form components. Finally, the information about how much time spent during the operation is displayed on the screen.

4.2.3 Making the Optimal Offloading Decision in Face Detection Scenario

At this step, we modified our face detection application with the ICEMobile framework to give the offloading capability. Our transformation on the application structure is depicted in Figure 4.2. The first and last nodes are not changed. However, in node B^l , there is a conversion to raw data with Base64 encoding to transmit the image data. Secondly, in node C^l , the necessary initializations for Socket, REST or RMI are completed. The node C^2 is the unique node that is executed on the cloudlet that makes the decoding operation and face detection operation. At the end of its execution, it sends the obtained results back to the mobile device.

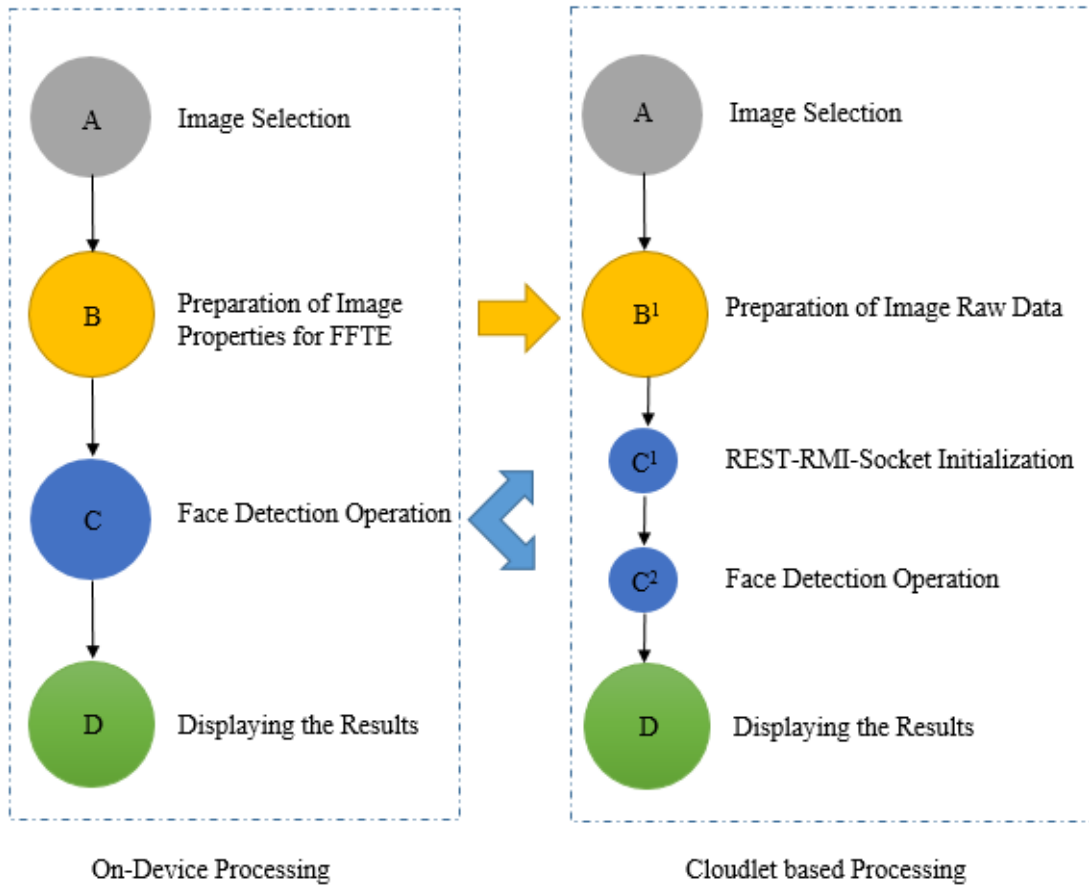


Figure 4.2: Call-Graph Transformation of Face Detection

According to the tests of on-device processing, the execution of the native Android FFTE method take more than 80% of the total energy consumption. Since it is a native function, we are unable to partition this function to obtain a more uniform distribution. As a result, we concluded that this case is not an ideal use case for optimization-based offloading. Figure 4.3 shows clearly that offloading will save energy for the images larger than 0.3M megapixels.

On the other hand, we used this scenario to make an analysis of different client/server communication techniques including Java sockets, RESTful Web Services and LipeRMI to find the most efficient offloading model.

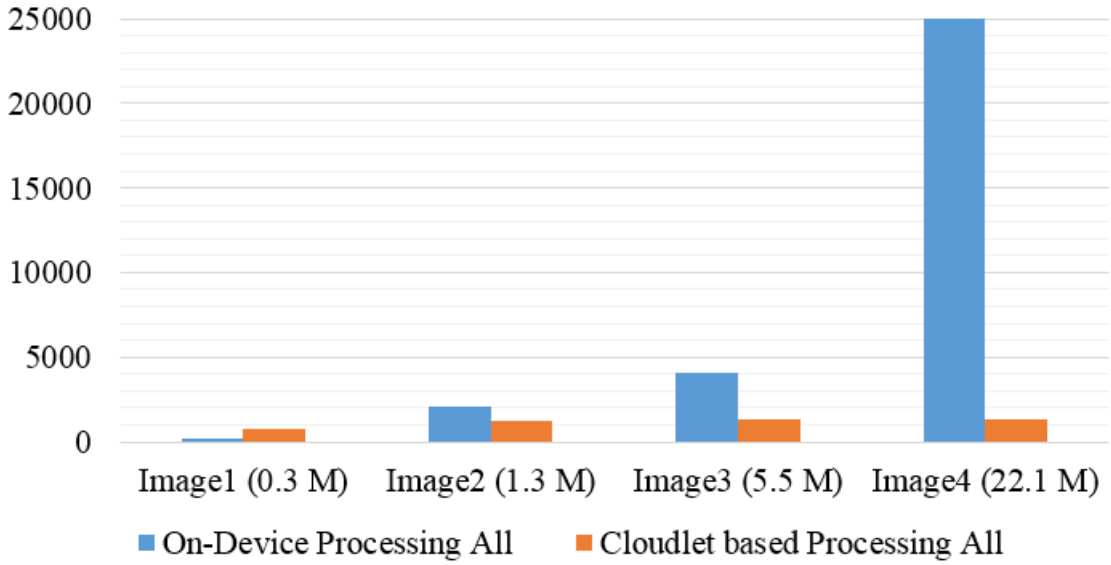


Figure 4.3: Comparison of Energy Consumption of Face Detection Operation

4.2.4 Comparison of REST, Socket and LipeRMI in Face Detection Scenario

As mentioned in Section 3.1, REST, socket and RMI based communications are amongst the most well-known offloading techniques. To compare the time and energy consumption of these communication types, we measured only the data transmission process of the use case, which starts by sending the data packet from client to server and receiving back from server. This process is illustrated as the edges $B^1 \rightarrow C^1$ and $C^2 \rightarrow D$ as shown in Figure 4.3. In order to obtain reliable results, we executed the operations for 10 times, and we calculated the average of measured values. As a result of our tests, as shown in Figure 4.4 and Figure 4.5, REST and Socket-based offloading consume nearly the same time and energy. The Lipe-RMI is found more costly than REST and Socket-based communications. However, Lipe-RMI has significant advantages for the ICEMobile offloading framework as stated in Section 3.

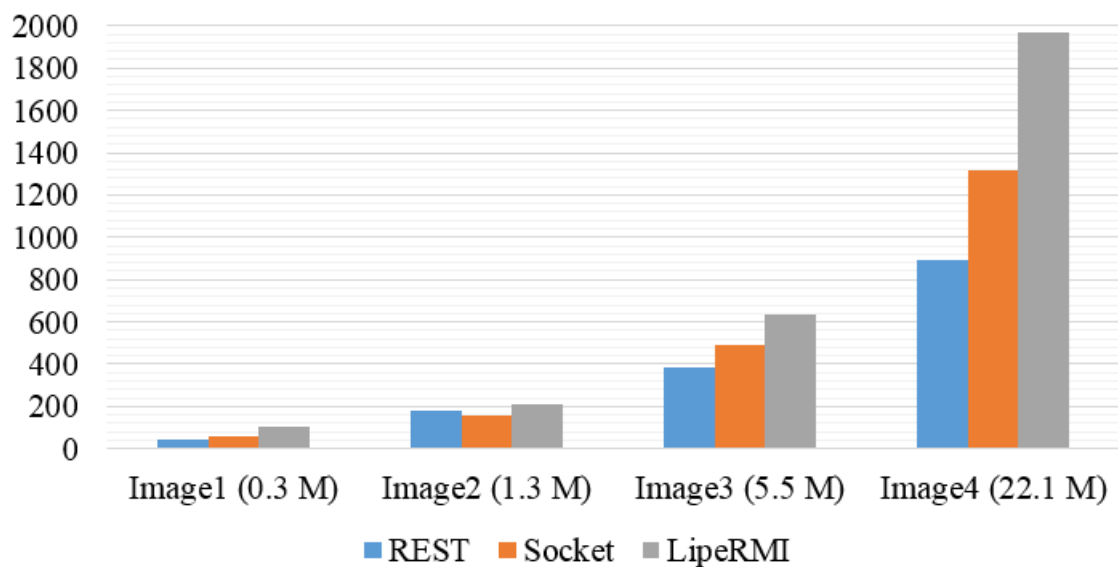


Figure 4.4: Comparison of Three Communications based on Time Consumption

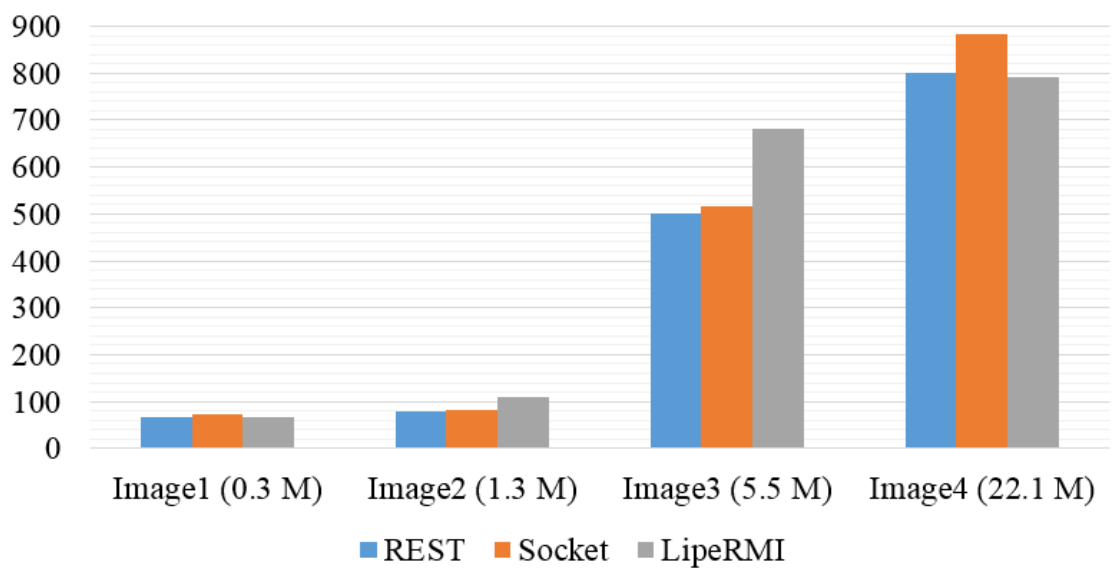


Figure 4.5: Comparison of Three Communications Based on Energy Consumption

4.3 OCR Scenario

In addition to the face detection use case, we implemented OCR scenario that is one of the mostly used computation intensive operations. Doing so, the aim is to extract the characters from a given image. The OCR is used in several places in daily life such as reading the signs written in a foreign language for a tourist, perceiving what is around for a blind person, etc. There is a need for the mobile cloud computing in these use cases, since they require real time responses to the mobile users, by executing computation intensive OCR operations. In the application markets such as Google Play and Apple Store, there are various applications that function as an assistant to the blind people that read the texts in printed documents using the mobile device camera. However, these applications consume huge amount of energy and their latency are still major problems.

In our OCR implementation, we used tess-two library in the mobile device and Tess4j in the cloudlet.^{19,20} They are both the implementation of the open source Tesseract OCR Engine that has been developed at HP Labs between 1985 and 1995, and it has been improved after the sponsorship of Google until today. It can read a wide variety of image formats and convert them to text in over 60 languages. This application is written in C and C++ languages. The architecture of Tesseract is shown in Figure 4.6.²¹ This procedure starts with an image selection and then ends with recognized characters.

¹⁹ Tess-two. <https://github.com/rmtheis/tess-two>

²⁰ Tess4j. <http://tess4j.sourceforge.net/>

²¹ <http://www.slideshare.net/oscon2007/os-raysmith>

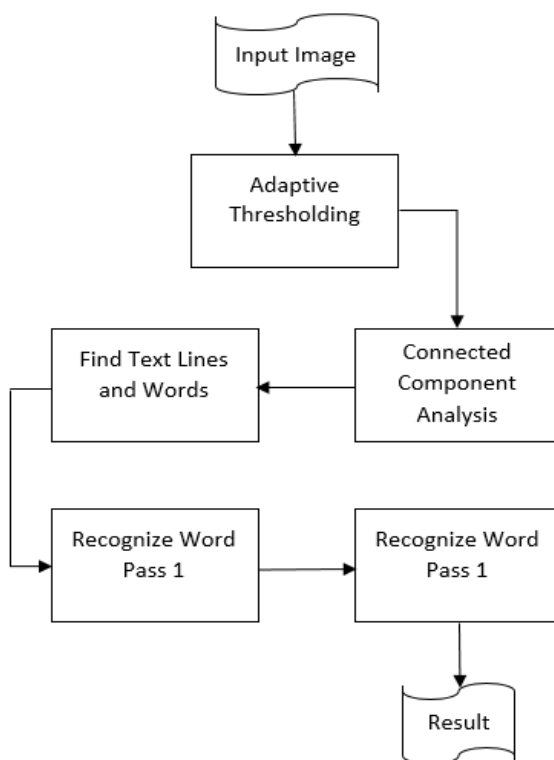


Figure 4.6: Tesseract OCR Engine Architecture

Figure 4.7 shows the screenshot of our application. As a similar interface with Face Detection application, the user selects an image containing characters and selects OCR function. The detailed time cost appears in the right side of the screen.

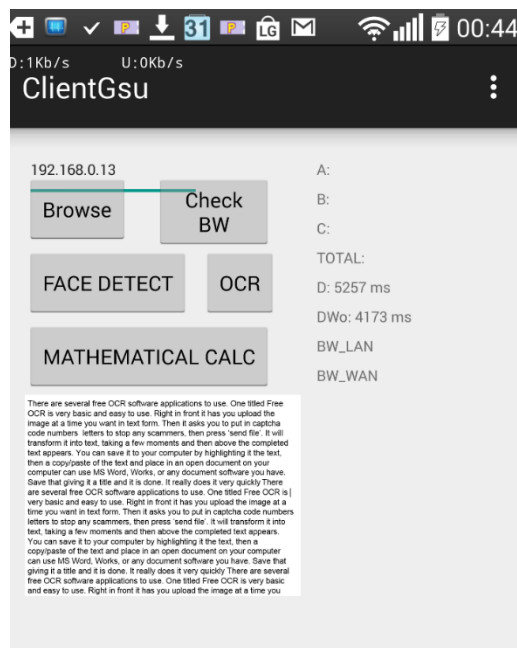


Figure 4.7: Screenshot of OCR Functionality of our Implementation

4.3.1 Preparing the Test Images for the OCR Scenario

According to Chang et al. (2005), the performance of the OCR operation is correlated with the number of characters in the given image. To make the performance comparisons, we generated text files with different number of English characters and we created image files for each text. Table 4.3 shows the properties of four image files that will be used in OCR tests.

Table 4.3: Selection of the Images Having Different Dimensions for OCR

	Number of Characters	Size on Disk
Image1	8	4KB
Image2	147	16KB
Image3	616	47KB
Image4	1188	88KB

4.3.2 Application Analysis of Android-Based OCR

The OCR application is composed of the following steps.

- a. **Image Selection:** The image containing characters is selected from the gallery. This is not an offloadable part of the application, since it needs to use Android view components.
- b. **Preparation of Image Properties:** The input image is converted into a bitmap object and the necessary data preparation for *nativeGetUTF8Text* is done.
- c. **OCR Operation:** *nativeGetUTF8Text* API is called with the given bitmap and the information containing the recognized characters is received.
- d. **Displaying the Results:** After the recognition operation, the application logs the obtained results.

4.3.3 Making the Optimal Offloading Decision in OCR Scenario

At this step, we modified our OCR application with the ICEMobile framework to give the offloading capability. The face detection and OCR are similar scenarios due to the fact that there is an image processing in each use case by using native functions. As a result, their call graph transformation realized as very similar to each other.

The call graph of OCR is valid with the call graph of Face Detection and can be applied with the same graph of Figure 4.3.

According to our tests of on-device processing, the execution of the native *nativeGetUTF8Text* operation takes nearly 90% of total time and energy cost. It is much higher than the face detection operation since there is not any painting operation on the image in OCR. Since Tesseract is not a Java-based library, we are unable to get into the structure of its API named as *nativeGetUTF8Text*, and it becomes unavailable to obtain a uniform distribution, with a similar issue we described in Section 4.2.3. As a result, there is no need to the optimization model. Figure 4.9 shows that offloading the OCR operation creates a time speed-up for each type of images. This result contradicts with the energy consumption values since offloading saves energy for images having more than 600 characters, as shown in Figure 4.9 shows. For the images having below 600 characters, the operation should be executed on the mobile device based on perspective of energy gain.

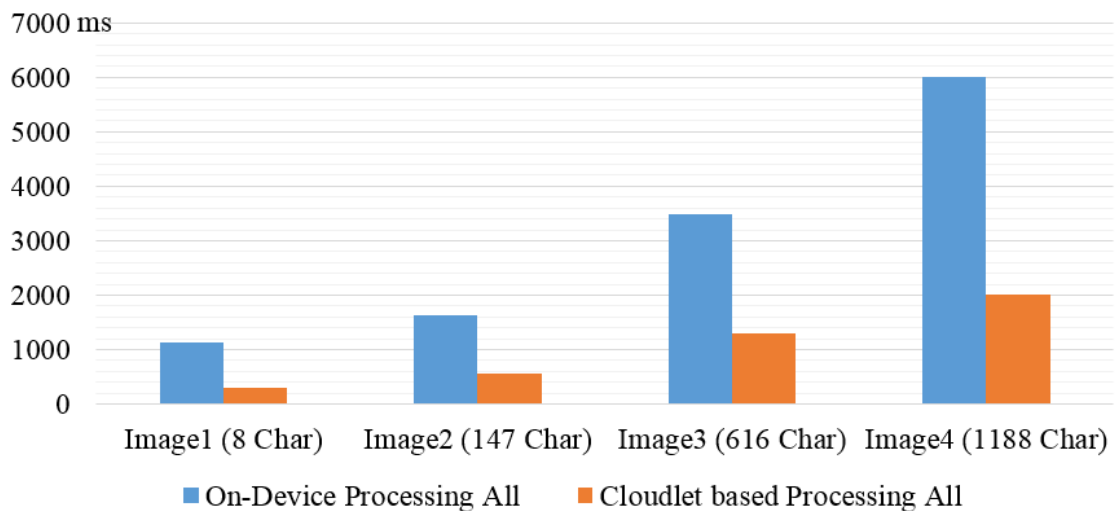


Figure 4.8: Comparison of Time Speed-Up of OCR Operation

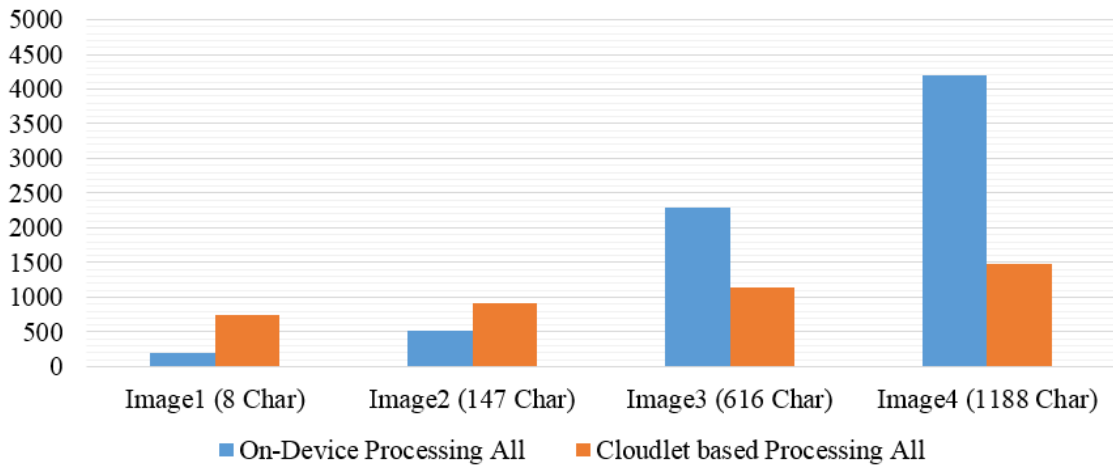


Figure 4.9: Comparison of Energy Consumption of OCR Operation

4.4 The Scenarios Consisting of Mathematical Calculations

In the previous sections, the implementations of the computation intensive scenarios of face detection and OCR showed that they are not convenient for optimization based offloading architectures since they contain native library implementations. Moreover on the mobile client level, there is not any possibility of dividing their functions into sub-parts. To resolve this issue, we created synthetic scenarios that are composed of matrix-based operations using JScience library. JScience is a Java library that is capable of realizing scientific calculations with its numerous APIs of economics, geography, mathematics and physics. The benefit of using JScience library is that its same version performs well on both mobile application and server software.

As a first trial, we performed a matrix multiplication operation of two matrices with size of 70x70 in both client and cloudlet. To make offloading possible, we used the Lipe-RMI framework, which is the main communication type of ICEMobile framework. In order to examine whether there will be a speed-up when this resource-intensive task is executed on the cloudlet, we realized the measurements of time consumption for each process. Figure 4.10 summarizes our approach of measuring time cost values for server-side and client-side processing in matrix multiplication operation.

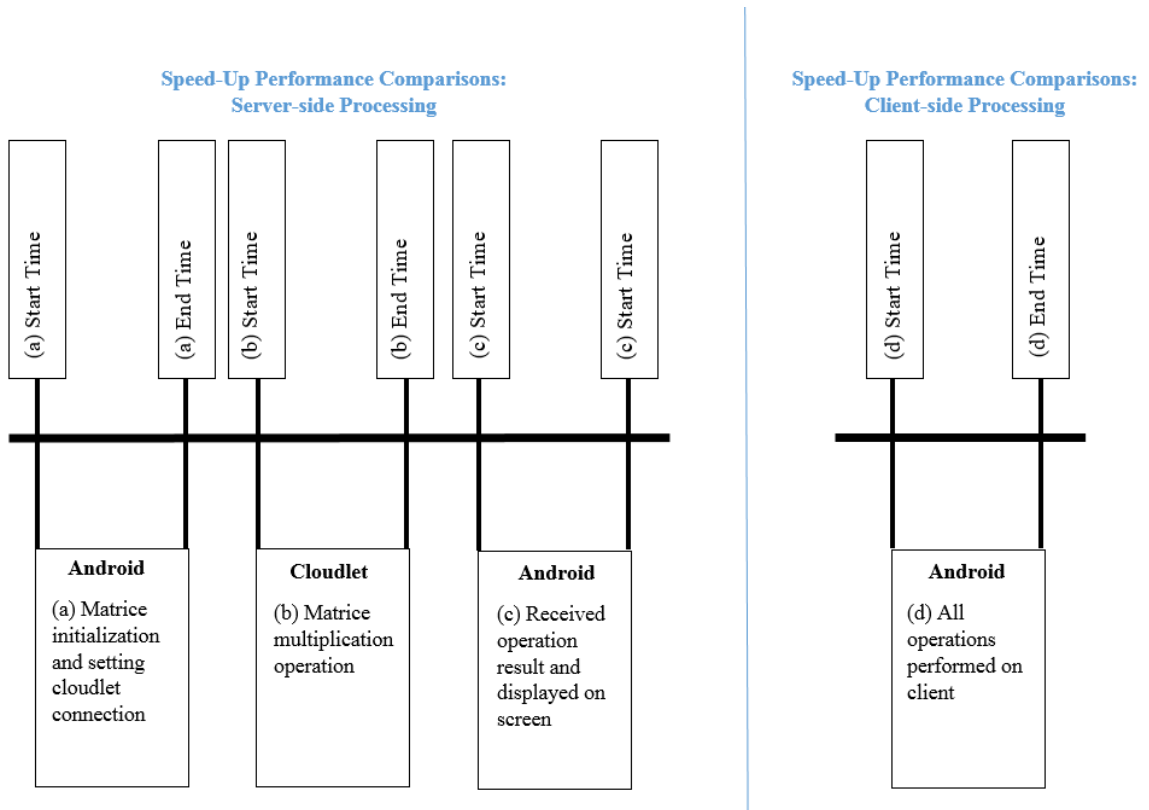


Figure 4.10: Comparison of Time Speed-Up of Server and Client-side Processing

In order to obtain more accurate values, we calculated an average value for each step. As a result of our tests, the execution of without offloading lasts 7x times longer than offloading, as shown in Table 4.4. The role of processes *a*, *b*, *c* and *d* are described in Figure 4.10.

Table 4.4: Performance Comparison of Matrix Multiplication (In milliseconds)

	a	b	c	Total Cost with Offloading (a+b+c)	Total Cost with No Offloading (d)
1	70	12	2	84	259
2	18	6	1	25	198
3	40	4	1	45	241
4	25	3	2	30	369
5	17	3	3	23	271
6	37	5	2	44	326
7	21	3	3	17	260
8	22	3	0	25	231
9	43	5	1	49	263
10	21	3	4	28	186
AVG	31,4	4,7	1,9	38	260,4

In this test, we illustrated the benefit of offloading by using a Java-based operation. In order to perform more detailed analysis, we added new matrix operations including matrix inversion, division and addition with several matrices having different size. Here, our aim is to create a scenario with the heterogeneously weighted nodes, which will provide building an optimization model that contains the offloading and not offloading decisions together.

4.4.1 Generating a Synthetic Call Graph

At this scenario, we implemented several matrix operations including matrix creation with random values, addition, multiplication, division and inverse operations at different size of matrices. The first node initiates the application flow after getting an input from the user. Figure 4.11 shows the call graph of these implementations. The vertical lines differentiate the row and column size of the matrices. The matrix creation and addition operations are relatively simple to be completed by the device.

On the other hand, the energy and time cost of multiplication, division and inverse operations are high, especially for the larger matrices. To create the variables and coefficients of the optimization function that is described in Section 3, we executed all of the nodes and edges and measured average time and energy costs for each component of graph. Their results are shown in Table 4.4 with the parameters stated in Section 3.2.3. In the Table 4.5, T_n^l represent the time that a node takes to execute the method locally, while T_n^r identifies the time that a node takes to execute the method remotely. $B_{m,n}$ describes the time that is required to transfer the necessary program state when m calls n . E_n^l shows the energy that is needed to execute the method locally for each node $n \in N$. $C_{m,n}$ represents the energy that is required to transfer the necessary program state when m calls n .

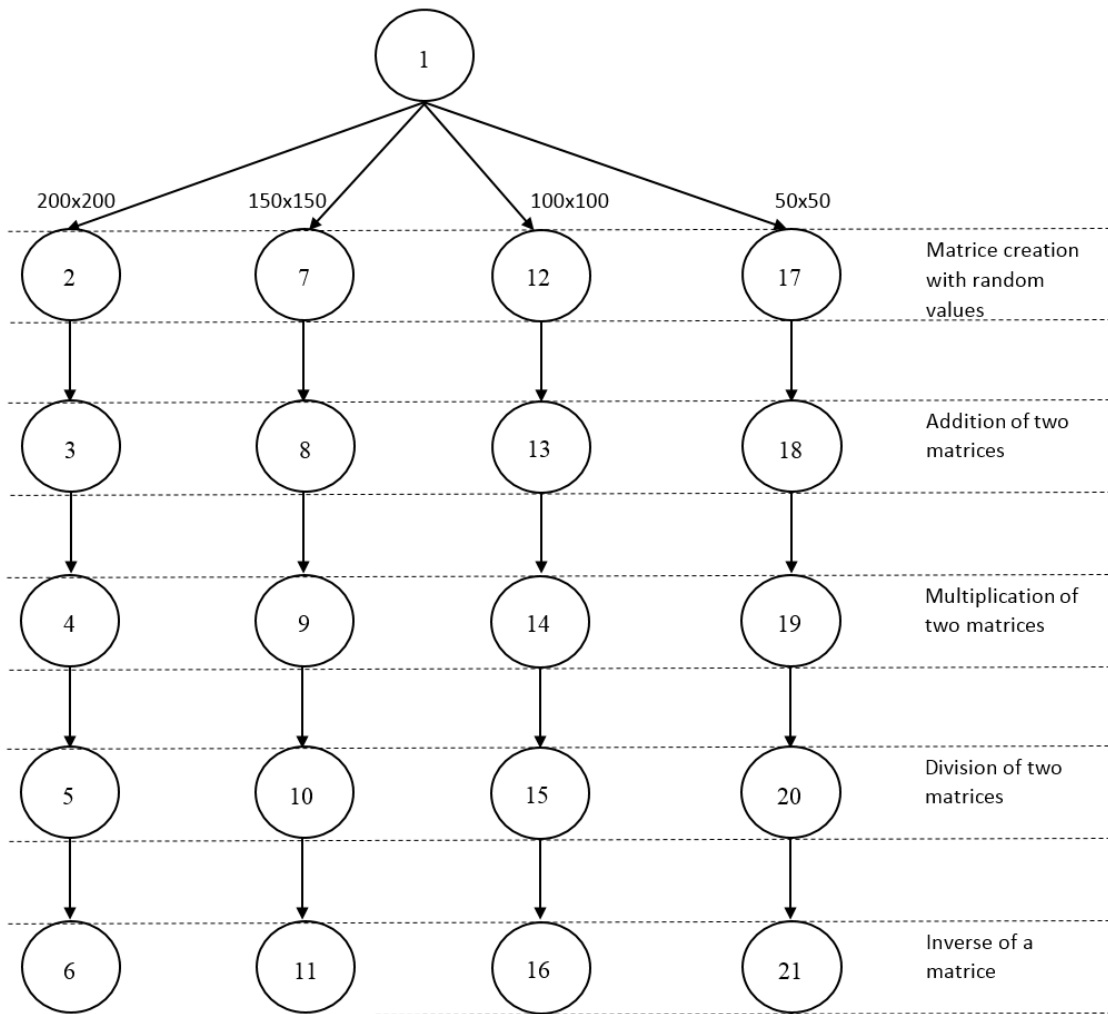


Figure 4.11: Call Graph of Matrix Operations

Table 4.5: Measured Time and Energy Costs of the Call Graph

node id	T_n^l	T_n^r	$B_{m,n}$	E_n^l	$C_{m,n}$
1	-	-	-	-	-
2	50	52	695	35,8	720
3	4	1	2020	2,8	680
4	328	25	2100	114,6	700
5	72699	700	2515	44556	700
6	73206	614	1832	42350	800
7	35	21	520	20	700
8	4	10	1010	7,9	700
9	162	15	1210	58,1	700
10	30260	284	1400	19200	700
11	29539	270	1170	15950	700
12	15	12	320	5	700
13	1	2	585	3,2	700
14	80	6	450	6,9	700
15	8643	89	575	4600	780
16	8889	89	685	4700	700
17	2	2	242	5,4	700
18	1	2	124	0,6	760
19	14	2	107	29,8	700
20	1107	14	128	801	700
21	1076	9	97	450	700

4.4.2 Results and Evaluation of Optimization in Synthetic Scenarios

We converted our weighted call graph into an optimization function. In the objective function and constraints, some of the variables have absolute values. Hence, it becomes a nonlinear program. We solved the problem as a 0-1 integer linear programming problem in R programming language, which is a powerful open-source statistical calculation engine. However, there is not any absolute value function in R language. To make it possible, we converted the absolute values into new constraints by assigning them to the newly defined variables. The optimization function has a strong focus on energy maximization of the mobile device. Somehow, it offloads the tasks even though it is more time costly to decrease the energy consumption of the mobile device.

We assumed that the cloudlet is an entity having unlimited energy and power capacity. This is the reason why we are discarding the energy consumption values of the cloudlet. The converted optimization function in R language contains 41 variables and 82 constraints. The 21 variables are the main variables that describe the offloading decision of the nodes and the remaining 20 variables are the temporary variables that are used for absolute value conversions. The most important constraint is the maximum time of the total operation, which prevents unexpected latencies caused by offloading. The maximum limit value is set to 105% of the total mobile processing time cost in case of there is not any offloading. This value identifies the maximum tolerable delay of the user. In addition to overall time cost constraint, there are also the constraints to identify the possible values of offloading for each node, with an integer value between 0 and 1. The remaining constraints are used for the absolute value conversions.

After making the absolute value conversions in the objective function and constraints (1), we define a new variable that keeps the total value of the absolute value (2). As a result of applying absolute value conversions, we obtain the equations shown in the equation (3) and (4).²²

$$|I_2 - I_3| \leq K \quad (1)$$

$$y_1 = |I_2 - I_3| \leq K \quad (2)$$

$$I_2 - I_3 - y_1 \leq 0 \quad (3)$$

$$-I_2 + I_3 - y_1 \leq 0 \quad (4)$$

²² Absolute values <http://lpsolve.sourceforge.net/5.5/absolute.htm>

4.4.3 Adjusting the Offloadability with Two Practices

In Section 3.2.3, we presented our optimization model and in the previous section, we defined our scenario with a weighted call graph. In order to execute the scenario, it is necessary to mark some of the nodes offloadable or not offloadable. This assignment is critical due to the fact the optimization function gives a decision based on the overall situations of the nodes. In the following scenarios, we observe and discuss the results of two practices, in which the offloadable nodes are different. In the first practice, named as “Unconstrained Offloading”, all of the nodes are marked as offloadable except the root node. In the second practice that is named as “Constrained Offloading”, we changed the offloadability of some nodes. The position of the nodes and their invocations have significant effect on the offloading decision. Moreover, another key parameter that affects the result of overall optimization function is the node offloadability. When a node is not offloadable, it probably needs a user input or accessing to a native component; i.e. it is obligatory to execute this node on the mobile device.

4.4.3.1 Unconstrained Offloading

We built this initial scenario by marking all the nodes as offloadable except the first node, which means there is not any node that interact with the native APIs. After making necessary operations to convert the optimization function into an integer linear programming, we executed it with the `lpsolve` function in R and we obtained the following result: Do not offload the first nodes in each vertical line of the graph, but offload all the rest. The interpretation of this result is that, the optimization function starts to offload with the lightweight nodes instead of the nodes having high energy cost while transferring. After transferring the node with lowest cost, the consecutive ones will not consume any energy since they are already on the cloudlet, so they will be operated there. This example shows that the utilized optimization function examines all of the nodes in high level, rather than considering it node by node. Since the set of mathematical operations are the same in each vertical with different size of matrices, the pattern is occurred in the same way. As a result, 16 of total 21 nodes (the black ones) are offloaded, as shown in Figure 4.12.

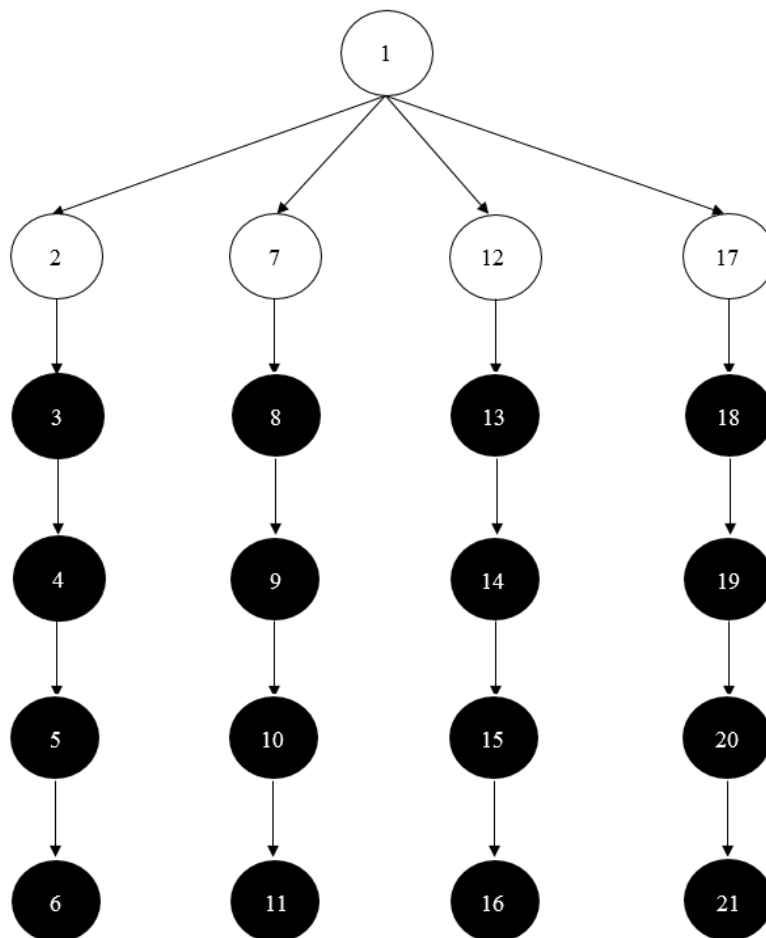


Figure 4.12: Call Graph of Unconstrained Offloading

4.4.3.2 Constrained Offloading

In the first scenario, entitled as “Unconstrained Offloading”, we found that more than half of the nodes are better to be offloaded in order to maximize energy efficiency. In order to discover the effect of node offloadability, we modified this scenario by marking the node numbers of 5, 10, 15 and 20 as not offloadable in addition to node 1. As a result of this execution, the offloading decision is given only for the final nodes in each vertical, as shown in Figure 4.13.

The interpretation of this situation is that when there is an offloadability constraint in the upcoming nodes x and the time and energy cost of current nodes are low, the optimization solver does not offload the nodes until the node with offload constraint is finished. The offloading decision of final nodes is another issue to be focused on. If their time and energy cost were lower on mobile processing, they were not be offloaded. But, their time and energy cost are not suitable to be executed on mobile, based on the perspective of having a total bounded time limit and target of energy maximization.

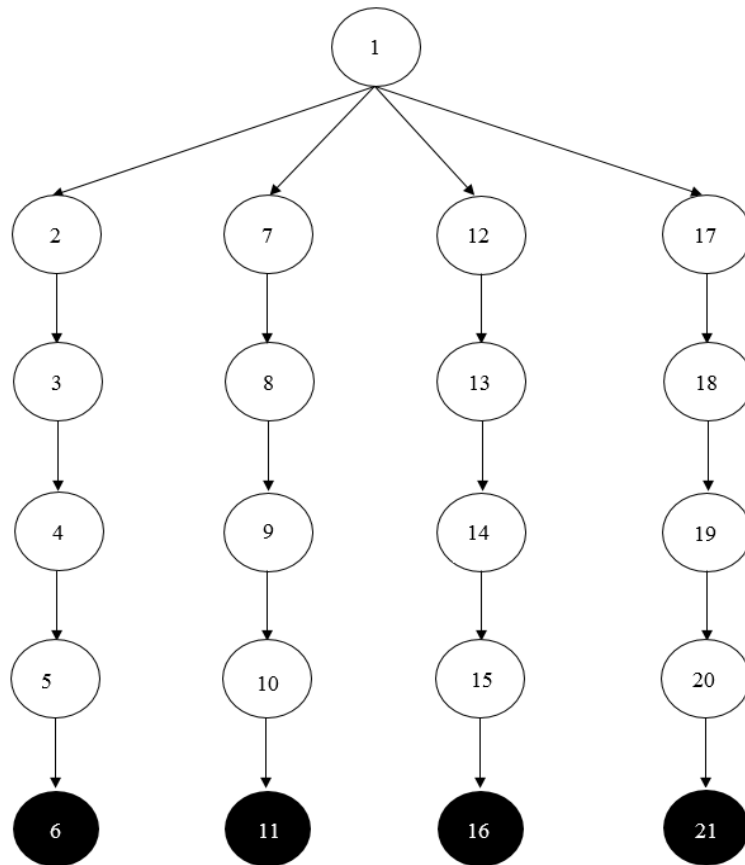


Figure 4.13: Call Graph of Constrained Offloading

As a result of the two scenarios, the obtained energy gains are summarized in Table 4.6. In ICEMobile enabled scenarios, it is possible to save energy up to 98% and 45% depending on the node offloadability constraints.

Table 4.6: Comparison of Obtained Energy Efficiency of the Scenarios

	Total Energy Cost (mJ)	The Energy Gain (mJ)	Percentage of Gain
Without Offloading	132.906	0	0 %
Unconstrained Offloading	2.874	130.032	98 %
Constrained Offloading	72.826	60.080	45 %

5. CONCLUSION

In this thesis, we presented the challenges in Mobile Cloud Computing and we mostly focused on the usage of nearby clouds entitled as cloudlets, in order to augment the execution capacity of the mobile devices. The advantage of using cloudlet services rather than the distant cloud services is the time and energy efficiency on the execution of computation-intensive tasks. In the literature review section, we compared different computation offloading techniques in respect to their architectures and capabilities. The proposed optimization based offloading method in one of the works, entitled as MAUI was the most similar approach to our perspective. Hence, we made use of their proposed optimization model. In this thesis, we implemented a lightweight RMI-based computation offloading in Java-based client and server and applied the optimization function in R statistical language. Initially, we made observations by implementing two real life use cases, face detection and OCR. We discovered that when the computation intensive part of mobile application is referenced to the native C or C++ based frameworks, there is not any need to the optimization based offloading, since it is not possible to measure the time and energy cost of their inner code blocks in Android runtime. On the other hand, by implementing a synthetic use case having Java based computation intensive operations, we generated a weighted call graph and solved a battery life maximization problem by converting the graph into an optimization function. In the call graph, the nodes describe the methods and the edges describe their invocations having time and energy weights. In order to measure their actual energy consumption in component level (CPU and Wi-Fi), we used the mobile application named as PowerTutor. As a result of the execution of the optimization function, we obtained the optimal offloading decision of the nodes and saved up to %98 energy consumption of mobile device. The ICEMobile framework will guide application developers modifying their applications to increase the mobile user experience and energy savings.

6. FUTURE WORK

In our research, the call graph generation is done as a manual process, which is not a suitable method when applying the framework into real-life mobile applications. One solution may be generating the call graph automatically based on application usage behavior of users. By implementing breadth-first or depth-first algorithms and realizing branch prediction, it is possible to create a call graph with the most commonly used functionalities. As a result, there will not be any need to manually generate the call graph. Moreover, our call graph generation component will provide a functionality that enables terminating call graph generation with the desired degree. It means that when the user gives terminating degree as “2”, the call graph generation will be done until the second method invocation for each node in the call graph. The benefits of this functionality is to become focused on the main functions rather than low details such as variable assignments, etc.

In addition to increasing call graph generation features, we also planned to extend our use cases with a Java based real-life applications. A possible scenario may be a face recognition application by using a convenient framework. The Java-based open source project, called as *Jama*, may be a strong alternative.²³ This framework contains a face recognition engine based on *Eigenfaces*. After selecting the appropriate framework, we are planning to execute a set of recognition operation with selected images and discover when the operations will be offloaded.

²³ Jama. <http://darnok.org/programming/face-recognition/>

REFERENCES

- Abunaser, A., Alshattnawi, S. (2013). Mobile cloud computing and other mobile technologies: Survey. *Journal of Mobile Multimedia*, 8(4), 241–252.
- Bakker A. (2014). Comparing energy profilers for Android. *21st Twente Student Conference on IT*.
- Buyya R., Yeo C., Venugopal S., Broberg J., Brandic I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616.
- Chang J., Balan R.K., Satyanarayanan M. (2005). Exploiting rich mobile environments. *Technical Report, Carnegie Mellon University*.
URL: <http://ra.adm.cs.cmu.edu/anon/2005/CMU-CS-05-199.pdf>
- Christensen J.H. (2009). Using RESTful web-services and cloud computing to create next generation mobile applications. *Proceeding of the 24th conference on Object oriented programming systems languages and applications - OOPSLA*, New York, USA: ACM Press, p. 627.
- Chun B. and Maniatis P. (2009). Augmented Smartphone Applications Through Clone Cloud Execution. *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Monte Verita, Switzerland.
- Cuervo E., Balasubramanian A., Cho D, Wolman A., Saroiu S., Chandra R., Bahl P. (2010). MAUI: Making Smartphones Last Longer with Code Offload, *Proceedings of the 8th ACM MobiSys*.
- Geohive (2015). Population of the entire world, yearly, 1950 – 2100
URL: http://www.geohive.com/earth/his_history3.aspx
URL: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- Kemp R., Palmer N., Kielmann T., Bal H. (2010). Cuckoo: a computation offloading framework for smartphones. *Proceedings of the Second International Conference on Mobile Computing, Applications, and Services, MobiCASE*.

- Kosta S., Aucinas A., Hui P., Mortier R., and Zhang X. (2012). ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” *Proceedings IEEE Infocom 12*, pp. 945– 953
- Mell P. and Grance T. (2011). The NIST definition of cloud computing US National Institute of Science and Technology.
- Research and Markets (2014). Mobile Cloud Market by Application, & By User - Worldwide Market Forecast and Analysis (2014 - 2019)
URL: http://www.researchandmarkets.com/research/7pj4cv/mobile_cloud
- Rountev A., Kagan S., and Gibas M. (2004). Static and dynamic analysis of call chains in Java. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1–11. ACM Press.
- Satyanarayanan M., Bahl P., Caceres R., Davies N. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):3
- Statista (2015). Global smartphone sales to end users 2007-2014
URL: <http://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/>
- Verbelen T., Simoens P., Turck F. De., Dhoedt B. (2012). Cloudlets: bringing the cloud to the mobile user, in: *Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing & Services*.
- Weiser, M. (1999). The computer for the 21st century. *Scientific American*, Vol. 265, pp. 94-104.
- Zhang X., S. Jeong, Kunjithapatham A., Gibbs S. (2010). Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. *Third International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications*, Chicago, IL, USA.

BIOGRAPHICAL SKETCH

Emre Çalışır was born in January 1, 1987 in Istanbul. He received his high school education in Burak Bora Anatolian High School. Furthermore, he received his Bachelor of Science in Computer Engineering from Galatasaray University in 2011. He works as a senior software engineer at Vodafone Research and Development Company.

PUBLICATIONS

- E.Çalışır, G.Isiklar Alptekin, A. Özgövde, “Yerel ve Hızlı Bulut Servisi: Bulutçuklar”, Akademik Bilişim, 2014, Mersin Turkey.
- G.K.Orman, O.Karadeli, E.Çalışır, “Overlapping Communities via k-Connected Ego Centered Groups”, IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 2015, Paris France.