ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTING

DEPARTMENT OF COMPUTING

A Lightweight Computation Offloading System for Mobile Cloud Computing

A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTING OF ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN SOFTWARE ENGINEERING

BY:

Alemybrah Tseghai

February, 2017

Adama, Ethiopia

Declaration

This thesis is my original work, and it has not been presented for a degree in any other university. All sources of materials used for the thesis have been acknowledged

_____

Alemybrah Tseghai

This thesis has been submitted for examination with my approval as university advisor

_____

Frezewd Lemma (PhD)

February, 2017

ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTING

DEPARTMENT OF COMPUTING

A Lightweight Computation Offloading System for Mobile Cloud Computing

By:

Alemybrah Tseghai

Names and Signature of Members of the Examining Board

Name	**Signature**	**Date**
Frezewd Lemma (PhD) Advisor	_____	_____
Endale Aragu (MSc) Chair person	_____	_____
Dereje Yohannes (PhD) External examiner	_____	_____
Mohd Wazih Ahmad (Asst. Prof.) Internal examiner	_____	_____
Mohammed Kemal (MSc) Department head	_____	_____
Dereje Regasa (MSc) School dean	_____	_____

ACKNOWLEDGMENT

First and foremost, I would like to thank my almighty God for making this possible. Next I would like to express my deepest gratitude and respect to my thesis advisor, **Dr. Frezewd Lemma** for his support and guidance throughout this thesis. His guidance and encouragement helped me a lot and put me on the right track. Finally, I would also like to thank my family and friends for the invaluable ideas and comments they provided me.

ABSTRACT

The latest advancements in mobile computing technology have changed human preferences for computing. Mobile devices are increasingly becoming essential and most effective computational tools. However, mobile devices are resource limited and those limitations prevent them from performing complicated tasks and running computation-intensive mobile applications. To overcome those limitations the computation offloading mechanism has been widely considered.

The existing computation offloading systems perform several tasks in the mobile side, including application profiling, application partitioning, obtaining and estimation of decision parameters, and offloading decision. Running those components (tasks) in the mobile side brings an extra computation overhead to the resource constrained mobile devices, which affects the performance gain from computation offloading. Moreover, computation offloading requires partitioning of the application into client and server parts for local and remote execution, respectively. A significant effort has been devoted to determine optimal partitioning techniques and to reduce the total execution time of the applications. However, most of the existing computation offloading systems haven't considered the intercommunication between the application parts during execution and the result of other solutions remain unsatisfying.

To alleviate those problems a lightweight computation offloading system that optimizes the performance of computation-intensive mobile applications is proposed in this research. The components of the proposed system that require heavy computation are implemented in the server side and a lighter offloading engine is designed to reduce overhead in the mobile device. In partitioning, the proposed system considers the call relationship between each method and the execution time of the methods. Instead of evaluating an individual method for offloading, it determines the offloading and integrating points for the whole application using the critical path method. Furthermore, the evaluation results of the proposed computation offloading system showed that it can speedup mobile applications 5.81X in average.

Key words: Computation Offloading, Improve Performance, Critical Path, Mobile Cloud Computing

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

MCC	Mobile Cloud Computing
GPS	Global Positioning System
ADT	Android Development Tools
IDE	Integrated Development Environment
BTS	Base Transceiver Station
RPC	Remote Procedure Call
RMI	Remote Method Invocation
PDA	Personal Digital Assistant
E-Learning	Electronics Learning
M-learning	Mobile Learning
WLAN	Wireless Local Area Network
MNO	Mobile Network Operator
SaaS	Software as a Service
IaaS	Infrastructure as a Service
API	Application Programming Interface
VM	Virtual Machine
XML	Extensible Markup Language
2G	Second Generation
3G	Third Generation
LTE	Long Term Evolution
WCDMA	Wideband Code Division Multiple Access
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
GPRS	General Packet Radio Service
CDMA	Code Division Multiple Access
DVM	Dalvik Virtual Machine
EC2	Elastic Compute Generation 2

JVM	Java Virtual Machine
RTT	Round Trip Time
ACK	Acknowledgement
SDK	Software Development Kit
MP	Megapixel
ms	Millisecond
Javassist	Java Programming Assistant
AT&T	American Telephone & Telegraph
M-healthcare	Mobile Healthcare
3D	Three Dimensional
m-game	Mobile Game

CHAPTER ONE: INTRODUCTION

1.1 Overview

Mobile devices such as smartphones, tablets and notebooks are increasingly becoming an essential part of human life as the most effective computational tools at anytime and anywhere[1]. However, mobile devices are restricted by their processor performance, bandwidth, storage capacity and battery lifetime[2]. These limitations prevent mobile devices from performing complicated tasks and running computation-intensive mobile applications. Some of these constraints are not due to the temporary backwardness of the current mobile hardware technology, but they are intrinsic to mobility[2].

Mobile cloud computing(MCC) is a new paradigm that uses cloud computing resources to overcome these limitations of mobile devices[3]. It is the combination of the services provided by cloud computing and mobile computing. The cloud environment is composed of several computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and accessed via the Internet[4]. Integrating mobile devices with the resourceful cloud environment provides a platform on which computation-intensive and feature-rich mobile applications are executed and delivered to the user efficiently. Moreover, the computation offloading technique is envisioned as a promising approach to coordinate this execution process which is the main focus of mobile cloud computing[5].

Computation offloading (aka mobile cloud offloading, cyber-foraging) refers to the technique used to migrate the whole or computation-intensive parts of the mobile applications to the servers in the cloud and receive the results back[6]. A significant amount of researches have been done on computation offloading mainly to minimize the energy consumption[7][8][9] and improve the performance[10][11] of mobile applications. Most of this computation offloading systems perform an extra computation to obtain the parameters involved in the computation offloading decision. Adding an extra computation such as application partitioning and execution time estimation on the mobile side can degrade the performance of mobile applications and affect the benefits of computation offloading[2].

The main aim of this thesis is to optimize the performance of computation-intensive mobile applications by performing application profiling and partitioning in the cloud environment at method-level granularity. Besides this our proposed computation offloading system determines the worth offloading point in the whole application using the critical path method instead of evaluating each individual method.

1.2 Statement of the Problem

In mobile cloud computing, computation offloading has been widely considered for minimizing the execution time (i.e. improving the performance) of mobile applications[12][10][11] and extending battery lifetime of mobile devices[13][9][7]. To achieve these goals the existing computation offloading systems perform several processing in the mobile side including application profiling, application partitioning, obtaining decision parameters and offloading decision. These processes(tasks) introduce an extra computation overhead to the resource constrained mobile devices, which affects the performance gain from computation offloading[14]. To alleviate this problem our proposed system performs application profiling and partitioning at the server side. Furthermore, a lightweight offloading decision engine is developed to reduce the computation overhead in the mobile side.

Moreover, most of the existing computation offloading systems[7][10][12] evaluate each application parts (i.e. Methods, objects, classes etc.) individually to determine whether the application parts are worth offloading or not. The intercommunication (call relationship) between those application parts are not considered. Separating application parts with higher call relationship in to the mobile device and the cloud environment causes large state transfer during execution[15]. A method call through the network creates a network overhead and a long communication delay, which affects the performance of mobile applications especially in a network with lower bandwidth.

Besides those, there are some other computation offloading systems that consider the communication between each application parts during partitioning. However, those systems lack an effective partitioning scheme that reduces the total execution time of the mobile applications. To overcome those problems our proposed computation offloading system considers the call relationship between each method and uses the critical path method to partition and reduce the total execution time of the mobile applications.

1.3 Objectives of the Study

1.3.1 General Objective

The general objective of this research is to develop a lightweight computation offloading system for improving the performance of mobile applications.

1.3.2 Specific Objectives

- Studying the computation offloading concepts and related topics in detail.
- Developing a lighter computation offloading engine for mobile cloud computing
- Developing a partitioning algorithm that partitions the mobile applications in the cloud using the critical path method.
- Developing an application profiler to obtain the execution time of methods at the cloud environment.
- Performing computation offloading at method-level granularity
- Considering the call relationship between methods during application partitioning

1.4 Research Specific Questions

1. How to optimize the performance of computation-intensive mobile applications?
2. Do considering the call relationship between the methods of mobile application enhance the performance of mobile applications?
3. How to minimize the computation overhead created by the computation offloading systems themselves on the mobile device side?
4. How to minimize the data transfer between the mobile device and the remote server during the execution of the mobile application?

1.5 Scope and Limitation of the Study

The proposed computation offloading system have the following aspects:

- ✓ Application partitioning
- ✓ Application profiler
- ✓ Implementing performance efficient computation offloading mechanisms
- ✓ Surrogate (i.e. remote server) discovery service
- ✓ A lightweight offloading decision engine
- ✓ The considering round-trip time during offloading decision

The proposed computation offloading system excludes the following features:

- ✓ Implementing battery efficient computation offloading techniques
- ✓ Server side resource utilization and optimization
- ✓ Implementation of the computation offloading system in public cloud environment
- ✓ Using the memory and CPU usage state of the mobile device as parameter for offloading decision

1.6 Methodology

Several research techniques and procedures have been considered to achieve the objectives of this research. The detailed description of these methods is presented as follows:

1.6.1 Research Design

This research focuses on improving the performance of computation-intensive mobile applications. Since the proposed computation offloading system is an improvement on the existing computation offloading systems, the design-oriented research methodology was used in this research. The design-oriented research methodology is a problem-solving paradigm that have five process steps[16], which was a natural method to solve this research problem. These five phases of design-oriented research methodology that were used in this thesis are problem identification, solution suggestion, development of the proposed system, evaluation and conclusion.

Problem Identification

At the very beginning of this research, existing computation offloading systems and techniques have been studied to identify their achievement and drawbacks. Based on the comparative analysis, the gaps of the existing computation offloading systems were identified. These problems are performing heavy computation at the mobile device side and using inefficient application partitioning scheme. The existing computation offloading systems perform massive computations to identify and offload the computation-intensive parts of mobile applications. Performing such a heavy computation in the resource constrained mobile device affects the performance gain from the computation offloading systems.

Solution Suggestion

In the second phase of this research, different solutions were determined and suggested for each research problems. The suggested solutions are:

- ✓ Deploying the heavy computing components of the proposed system in the cloud environment.
- ✓ Developing a new application partitioning technique that determines the integration and offloading points in the applications using the critical path method.
- ✓ Developing a lighter offloading decision engine for the mobile devices.

Development of the Proposed System

A lightweight computation offloading system is developed to improve the performance of mobile applications. The computing requirement of every components of the system was studied and the components that require heavy computing have identified. The application profiler and the application partitioning engine (which are computing intensive) are deployed at the server side. Whereas a simple and lighter offloading decision engine is developed for the client side to reduce the computation overhead. Moreover, the proposed system offloads the application at the method-level granularity and the RPC technique was used for communication between the client (i.e. Mobile device) and the servers in the cloud environment.

Evaluation and Improvement

Eventually, the proposed computation offloading system was implemented and evaluated. The cloudlet computation offloading architecture was used to implement and evaluate the proposed system. A mobile application that uses image processing algorithms to apply some effects has been integrated and used to evaluate the proposed computation offloading system.

Discussion and Conclusion

The results obtained in the evaluation phase and the overall achievement of the proposed computation offloading system are discussed in detail. The performance achievement obtained in this research was compared with the achievement of prior computation offloading systems. Finally, the findings of the research are summarized.

1.6.2 Tools and Techniques

Microsoft office word has been used for preparing the document and Microsoft Visio was used for designing. Eclipse neon with Android Development Tools(ADT) was used to develop the proposed system. The Eclipse development environment was chosen over the Android studio because of the following benefits:

- ✓ Easy to develop and customize mobile applications
- ✓ Easy to navigate
- ✓ Several helpful plugins are available

Since the Android operating system is java dependent, we used java programming language to develop the proposed computation offloading system.

1.7 Significance of the Study

The major significance of the proposed computation offloading system is enhancing the performance of computation-intensive and feature-rich mobile applications. Mobile applications are becoming increasingly more complicated and computing intensive[17]. It takes long time or impossible at all to run these applications in the resource constrained mobile devices. Computation offloading is an attractive solution for shortening response time (i.e. improving the performance) of such complex mobile applications[18].

It is more essential for mobile applications that obtain several input data from different sources and perform processing at real-time. For example, context-aware mobile applications receive input data from different sources like GPS, maps, accelerometers, temperature sensors, etc., those data need to be analyzed together in order to obtain real-time information about users' context[18]. A slow response of such real-time applications affect the accuracy of the result (in this case it may provide a wrong users' context). These real-time applications require higher speed processor to perform these tasks at a time, which is the main limitation of mobile devices. A computation offloading system which highly optimizes the performance of the applications is the only way to meet such time-constraint.

1.8 Organization of the Thesis

The rest of this thesis is organized as follows, Chapter Two covers literature review. Chapter Three presents the existing works that are related to the objective of this thesis. The design of the proposed computation offloading system is discussed in Chapter Four followed by implementation and evaluation in Chapter Five. Finally, the conclusion made on the thesis result and future work of this thesis are presented in Chapter Six.

CHAPTER TWO: LITERATURE REVIEW

2.1 Mobile Cloud Computing (MCC)

Nowadays, a user either relies on the mobile or the cloud to perform various daily life activities, e.g., social communication, video streaming, gaming, banking, learning etc.[3]. Mobile Computing is a technology that allows transmission of data through a computer or any other wireless enabled device without having to be connected to a fixed physical link[19]. The emergence of portable devices such as personal digital assistants (PDA), Tablets and Smartphones has made mobile computing very convenient. The portability of the devices ensures and enables user to access all computing services easily, at anytime and anywhere.

On the other hand, Cloud computing is a model for enabling ubiquitous, convenient on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[12]. Cloud computing promises the ubiquity and availability of virtually infinite resources and provides a utility computing model, where consumers pay on the basis of their usage (pay- as-you-go)[3]. Mobile Cloud Computing (MCC) has emerged as the combination of those two promising fields. The cloud provides a ubiquitous computational and storage platform to process complicated tasks, while the smartphone grants the mobility features to process simple tasks[3].

“Mobile cloud computing refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers”[20]. It is the aggregation of the rapidly growing mobile computing and the highly promising trend called cloud computing. The integration of mobile technology to cloud computing helps to overcome the limitations of mobile devices and provides computing services in almost unrestricted mobility.

Figure 2.1 The general idea of mobile cloud computing (MCC)[17]

2.1.1 Mobile Cloud Computing (MCC) Architecture

The general architecture of mobile cloud computing is depicted in Figure 2.2 the components of mobile cloud computing architecture and their functions are discussed as follows:

a. Mobile Devices

Mobile devices are connected to the network through base stations that establish and control the connections (air interface) and functional interfaces between the networks and mobile devices. These base stations can be Base Transceiver Station (BTS), access Point or Satellite[21].

b. **Network Operator**

It has Base Station through which mobile devices are connected. These Base Stations are connected to the central processors which process the mobile user's request. Mobile users obtain AAA (Authentication, Authorization and Accounting) service from the mobile network operator.

Mobile users' requests and information are transmitted to the cloud controllers that are connected to servers providing mobile network services. The subscribers' requests are delivered to a cloud through the Internet[22].

c. **Cloud Controllers**

The cloud controller receives the subscribers' requests from the network operator via the Internet and process the requests to provide a corresponding cloud services to the mobile users. It uses the resources in the datacenter to provide different services for mobile users[21]. These services are developed and provided with the concepts of utility computing, virtualization, and service-oriented architecture (e.g., web, application, and database servers)[22].

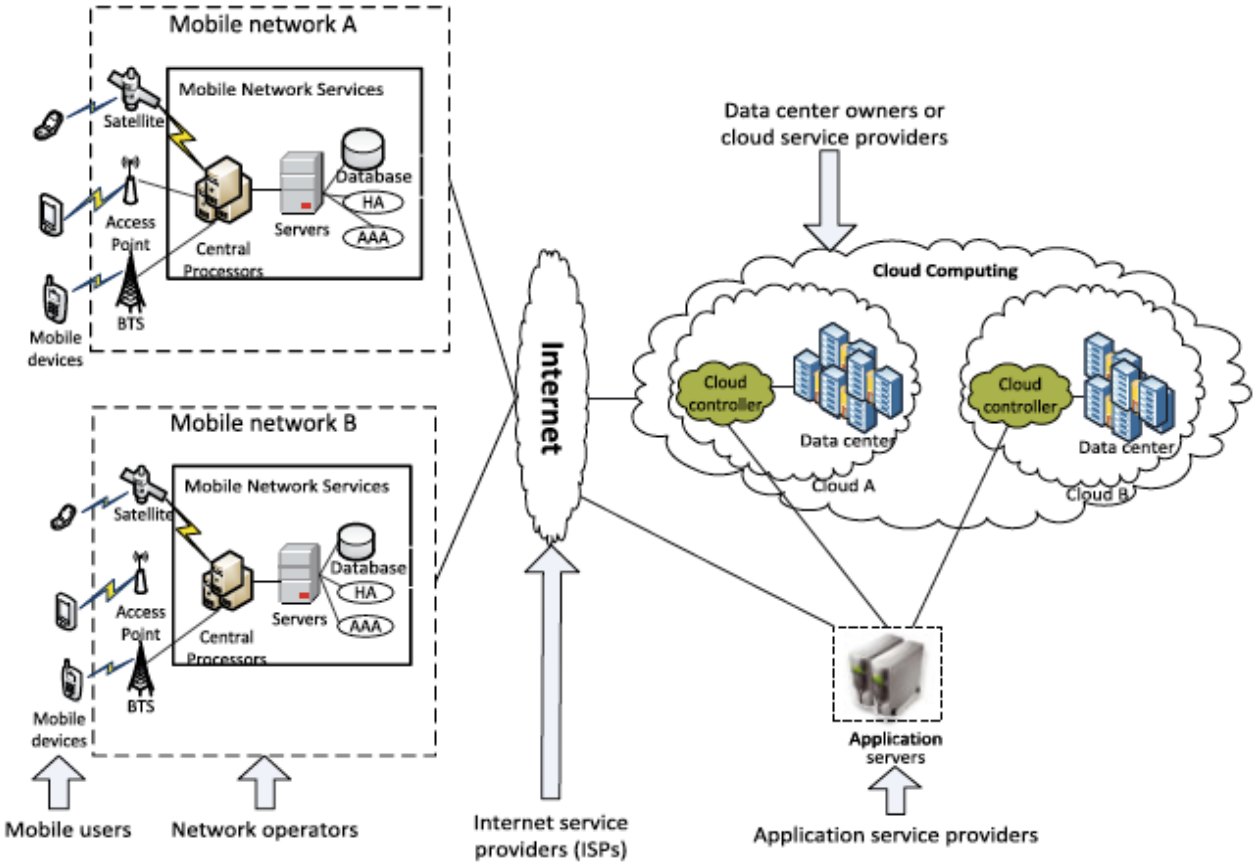

Figure 2.2 Mobile Cloud Computing (MCC) Architecture[22]

2.1.2 Mobile Cloud Computing Applications

a. Mobile Commerce

Mobile commerce (m-commerce) provides business models for commerce using mobile devices such as mobile banking, mobile advertising, mobile shopping etc. M-commerce applications require scalable processing power and security measures to accommodate a high volume of traffic due to simultaneous user access and sensitive data transaction processing. These requirements can easy be fulfilled by integrating the m-commerce applications with cloud environment[22].

b. Mobile Learning

M-learning is the delivery of e-learning through the mobile device. Traditional m-learning has limitations such as high cost of devices/network, low transmission rate, limited storage of educational resources[22]. The cloud-based m-learning can alleviate these limitations. Cloud computing can store a huge amount of educational resources and provide infrastructure, platform, and application services for users and it can also provide unlimited computing power for the completion of various types of application[23]. A centralized cloud-based m-learning can enhance the communication quality between students and teachers. It can also create a conducive environment for collaborative learning.

c. Mobile Healthcare

It is a medical and public health practice delivered through mobile devices to minimize the limitations of traditional medical treatments. M-healthcare applications can be used for chronic disease management, personal health record, health information delivery, remote patient monitoring and so on. Cloud oriented mobile healthcare allows a large amount of patient data to be stored on the cloud. This makes the delivery of healthcare service reliable, scalable and highly accessible[22].

d. Mobile Gaming

Nowadays, the mobile gaming scenario includes a wide range of games (from very simple graphic games to cutting edge 3D graphics, from single player to multi players games)[24]. These complex and computation extensive m-game application require high computing resources which can be beyond the capacity of the mobile devices. The storage and processing of such

computation extensive applications at a remote server in the cloud can be used to overcome these limitations.

e. Other Applications

Mobile cloud computing can also be used to enhance assistive technologies for disabled individuals. For example, it can be used to transform the pedestrian crossing guide and mobile currency reader for blind and visually-impaired individuals. It can also enhance the functionality of multimedia Sharing, Keyword-based, voice-based, tag-based searching, smart home systems, etc.[22].

2.1.3 Advantages of Mobile Cloud Computing

i. **Improving data storage capacity and processing power:** since, the data storage and processing is done at the remote server in the cloud, there will be a significant improvement in storage and processing capacity in the resourceful servers, compared to the resource constrained mobile devices.

ii. **Improving reliability and availability:** data stored in mobile devices can be lost due to the lost or damage of the mobile device. Storing the data in the cloud helps to access the data of the lost or damaged device using different mobile device.

iii. **Easy content and resource sharing:** data and applications stored in the cloud environment, can be accessed and shared by mobile users at any time. Cloud service providers facilitates the content and resource sharing among authorized users or customers.

iv. **Extending battery lifetime:** reducing the processor load can significantly minimize the energy consumption of a device[22]. There is a technique called computation offloading in mobile cloud computing, which minimizes a processor load in mobile device. This technique migrates heavy computations and complex processing from the resource-limited mobile devices to the resourceful servers in the cloud environment.

2.2 Mobile Computation Offloading

Mobile devices, such as smart phones and tablets have become the most demanding computing devices due to their ubiquity. These devices provide a wide variety of useful services (e.g., web surfing, video calls, emails, gaming, navigation and guidance, healthcare, banking, education etc.) from anywhere at any time. However, mobile devices are restricted by their processor

performance, bandwidth, storage capacity and battery lifetime[2]. These constraints handicap mobile devices from performing complicated tasks and running computation-intensive mobile applications. A prominent solution to overcome these limitations is computation offloading, which is the focus of mobile cloud computing (MCC)[10].

Computational offloading refers to a technique, in which a computational-intensive application parts are intercepted from a local execution workflow and migrated to a remote server in the cloud environment for execution, and the result of the processing is synchronized back into the local work flow[3]. Computation Offloading can be performed at different levels of granularity. It can be performed at the whole application level, class level, method level, thread level, object level etc. Moreover, computation offloading is different from the traditional client-server architecture where a thin client always migrates computation to a remote server[25]. Whereas in computation offloading, a computation-intensive part of an application is migrated to a remote server only when offloading is worthwhile.

2.2.1 The Reasons to Perform Computation Offloading

There are several reasons to perform computation offloading, the following are some of these reasons:

- ✓ Mobile applications are getting more complicated and more computing intensive, which brought a mismatch between requirements and the mobile devices' capabilities[17].
- ✓ Mobile users require shorter response time when they use applications in mobile devices with inherent limitations[17].
- ✓ Users would like to extend the battery life time of their mobile devices.

2.2.2 The Steps to Perform Computation Offloading

Even though the steps to perform computation offloading depends on the structure and type of computation offloading systems, the mostly used sequence of performing computation offloading is presented in[17], which are as follows:

1. Searching for available surrogates and services, in which mobile devices can offload their computing intensive tasks to.
2. Monitoring environmental context information such as status of network quality and relevant devices, and predicting required values.

3. Performing application partitioning at a suitable granularity and making offloading decision.
4. Remote execution control, including how to interact with mobile devices, such as getting users' input and returning results to mobile devices.

2.2.3 Computation Offloading Architectures of MCC

Computation offloading architecture describes the type of resources used to optimize the ability of mobile devices and the way in which mobile devices are connected to and interact with cloud resources. Based on the cloud resource organization, existing computation offloading architectures can be classified in to the following four classes[17]:

i. Public cloud

The whole or portions of the mobile application is offloaded to public computing resources comprised of stationary servers through the Internet[17]. Some offloading frameworks[7][9]use this architecture to optimize the performance of the applications and to reduce the energy consumption of the mobile device. The disparity in computing capabilities between the cluster of server's in public cloud and smart phones is high and persistent. Public cloud is highly available, scalable, and resource elastic with powerful computing capabilities while mobile phones have limited resources[17]. Therefore, leveraging public cloud to execute mobile device tasks can highly enhance the performance of the mobile application. But the propagation time and communication cost must be considered for offloading because access to the public cloud may incur higher delay due to the lower quality of the cellular network.

ii. Cloudlet

A Cloudlet is a resourceful computer or cluster of computers that is well connected to the public cloud via Internet and available to nearby mobile devices mainly through Wi-fi[26]. Due to the high speed of the wireless LAN and its proximity, it minimizes the response time of the applications[26]. Minimizing the response time is important for intensive and sophisticated real-time mobile applications which are the main candidates of computation offloading. The Cloud-Vision[27] is one of the frameworks that offload mobile applications to nearby Cloudlet to minimize the response time. Even though offloading to nearby cloudlets provides a better performance, it also faces some practical problems such as reliability issues and restrictions in mobility.

iii. Cloud of Mobile Network Operator (MNO)

A mobile network operator is a provider of wireless communications (services) for mobile devices. It owns or controls all of the items necessary to sell and deliver services to an end user such as radio spectrum allocation, wireless network infrastructures, and provisioning of computer systems and marketing. By implementing the cloud concepts including SaaS, IaaS, MNO can be a cloud provider. Furthermore, MNO can be a cloud broker which plays as a service mediator between other cloud providers and mobile users. There are several MNOs that provide cloud services such as the China Mobile and AT&T. Cloud of MNOs covers a large area, its more safe and reliable[17].

iv. Cloud of Mobile Devices

In the other three architectures, the powerful computers in the cloud are the computing resource providers to the resource constrained mobile devices. But there might be a situation in which an access to those resources is not available and/or it is too expensive to access them. A virtual cloud formed from the collection of mobile devices in vicinity and in stable form can be a solution for such situations[17]. The virtual cloud provides computing resources and processes the offloaded mobile applications. Hyrax[28] introduced a concept of using the collection of mobile devices as computing resource providers. With cloud of mobile devices, users can share resources and execute tasks among the devices to gain better experience and performance[17]. However, a fluctuation of computing power might happen due to the mobility of the mobile devices, which affects the performance of the computation offloading system.

2.2.4 Communication Models

a. Client-Server Model

Several existing computation offloading systems have used the client-server communication model[29][30]. The model uses Remote Procedure Calls (RPC) or Remote Method Invocation(RMI) technology for communication between the mobile device and the remote servers[31]. Those well supported APIs provide the ability to make remote calls. A client instance and a server platform developed using those API are required to enable offloading. The client instance partitions the application and identifies the worth offloading (intensive) application parts. The server platform invokes the application parts that are identified as worth offloading and executes them.

b. Virtual Machine Migration

Virtual Machine migration is the process of transferring the memory copy of a virtual machine to the destination server without interrupting its process[31]. Since the offloading is done at the operating system level, this approach does not require the modification of the applications code. The VM limit protects the surrounding mobile devices which helps to provide a generally safe execution environment. The drawback of this approach for offloading is the overhead encountered due to the migration of the entire virtual machine (VM) from the mobile device to the server. In addition, the server must already be running a hardware simulator with the same configuration as the mobile device, otherwise it is complex to synchronize the VM in the mobile device with the VM in the remote server.

c. Mobile Agents

Mobile agents are autonomous programs that can freely move from a device to another device in a heterogenous network[25]. Thus, mobile agents can be used to facilitate the migration of tasks from mobile devices to the cloud environment. The platform independent languages like Java and XML have introduced a conducive infrastructure for mobile agents. Therefore, mobile agents can be migrated across heterogeneous platforms, which is suitable for computation offloading[32]. The scavenger[33] framework employs cyber-foraging (computation offloading) using Wi-Fi for connectivity, and uses a mobile agent approach to partition and distribute tasks.

2.2.5 Application Partitioning

The application partitioning is the most important phase of computation offloading, which is the process of determining which parts of an application are worth offloading and which are not[34]. It mainly focuses on splitting the execution of the application into the mobile device and the servers in the cloud environment, so as to minimize the total execution cost. Through partitioning, a mobile device can benefit most from offloading[35]. Applications can be partitioned statically during the development or dynamically during execution.

2.2.5.1 Application Partitioning Granularity

The first step in partitioning an application should be determining the right granularity, i.e. the extent to which the application is decomposed into smaller components. Computation offloading issues such as compatibility, object identity, class unloading, state transfer and performance

overhead depends on the different granularity levels of application partitioning[17]. Different researches on computation offloading have been done at various granularity levels, which are:

- ✓ Thread level: a thread level partitioning and offloading the thread to the cloud[7][36][37].
- ✓ Method level: intensive methods are identified and offloaded to the remote servers[9][38][39].
- ✓ Task level: some tasks or processes that have higher processing time are offloaded to the cloud[40][41].
- ✓ Object level: an entire object is offloaded to a cloud environment[42].
- ✓ Class level: objects of the same class are offloaded to the same server[43].
- ✓ Module level: an entire module or bundle is offloaded to remote servers[44].
- ✓ Whole application: the entire application is offloaded to remote environment[45].

These different granularity levels can be grouped into two groups:

a. The Coarse-Grained Systems

These systems consider the higher level of granularities such as module level or entire application level[34]. Since the application is partitioned into few and large components, it's simple and have a lighter overhead in the mobile device but it may increase data transmission which can highly affect the performance of the system, especially in a poor network.

b. Fine-Grained Systems

This approach considers offloading at a finer granularity level such as thread level, object or method level to outsource only the sub-parts that benefit from remote execution. This approach requires intensive monitoring and synchronization mechanisms on both the mobile devices and the cloud nodes during execution. Furthermore, its more challenging to ensure consistency in the distributed executions of mobile applications at finer granularity level[17].

2.2.5.2 Application Partitioning Methods

There are two ways of partitioning mobile applications for computation offloading, which are manual or automated partitioning.

a. Manual Partitioning

The computation expensive application parts are identified by the developers. This is said to be efficient, as it is supposed that the developer is aware of the intensive parts of the applications. However, the computation intensiveness (execution time) of a program mainly depends on the input parameters which makes hard to manually identify the computation intensive parts of the application. Furthermore, performing the partitioning can be a burden to the developer, especially if offloading techniques were not considered at the beginning.

b. Automatic Partitioning

The partitioning is done by a system that uses dynamic program analysis and/or static program analysis. it considers different input parameters and the structures of the application for partitioning. This makes the system more accurate and worthwhile. However, the computation for determining the partitions automatically is very expensive due to the computation required to analyze the program of the application.

2.2.6 Decision Making for Offloading

Since computation offloading migrates computation to a more resourceful computer, it involves making a decision regarding whether and what application part to migrate. It is not effective to offload the whole application to the remote server[46]. It is mainly due to the high communication delay that may be generated and some tasks require or are related to local features (sensors, input/output devices, user interfaces etc.)[2]. it's more advantageous to execute such tasks or parts of the application in the local mobile device. The effectiveness and benefits of offloading is highly depended on whether it is made at the right time, in the right way.

Offloading decisions are different from each other in decision objectives and decision approach they use. A significant difference is that an offloading decision can be made statically before the execution of the applications or dynamically during runtime[17]. Static decision has the advantage of low overhead during execution; however, this approach is valid only when the parameters can be accurately predicted in advance[25]. Dynamic decisions use different algorithms to adapt the context of the run-time environment, such as the fluctuation of network quality, the execution cost and the communication cost between each component or application parts. Mostly dynamic decisions cause higher overhead due to the complex computation required to monitor the run-time conditions and parameters[25].

There are different deciding factors in determining whether computation offloading would be beneficial or not[47]. Factors such as, the input data, the volume of the execution state transfer, the quality of the network, and the characteristics of the application and the mobile device might cause a variation. Computation offloading is almost always beneficial in a network with high-bandwidth but on slower networks, the offloading cost-benefit analysis is not as clear cut[47]. In general, the effectiveness of an offloading system is determined by its ability to infer where the execution of the code (local or remote) creates less computational effort for the mobile, such that by deciding what, when, where, and how to offload correctly, the device obtains a benefit[46].

i. What to Offload?

Offloading decisions should determine what types of mobile applications and application parts are worth offloading. Applications and computation tasks that are computation intensive and that requires low data transfer (execution state transfer) are more suitable and worth offloading. Other than these there are many applications and computation tasks that are difficult to be classified as suitable for offloading or local processing. Offloading the whole mobile application to the cloud is not always necessary or possible, due to the high communication delay that may be generated or because some tasks access local features during execution (sensors, user interface, etc.). It is more beneficial to partition the application and determine the tasks to be offloaded by considering the balance between how much the offloading saves and how much extra cost can be incurred[2].

ii. Where to Offload?

As its explained above in the computation offloading architecture, the partitioned application parts can be executed on multiple application processing nodes such as local device, cloud of mobile devices, nearby cloudlet and servers in public cloud[17]. Depending on the computational requirement and limitations, offloading decisions should include where to execute the offloaded parts of the application. The components can be executed locally on the mobile device or they can be offloaded to a nearby cloudlet or they can be offloaded and executed at the remote servers in the public cloud[2].

iii. How to Offload?

The mobile device uses several types of wireless networks for offloading. It can be performed using a cellular connection (e.g. 2G, 3G or LTE) or via an intermittently available WLAN

hotspot[12]. A weak wireless network can increase the response time and raise power consumption[2]. Offloading decision should include selecting the communication channel based on the network quality and on how to offload tasks through different and variable wireless channels.

iv. When to Offload?

Offloading is beneficial when the estimated execution time of the task on the mobile device is greater than the sum of its estimated execution time on the cloud server and the propagation time of the data. Offloading decision should determine when offloading a computational task to the dedicated remote server is optimal and when, on the contrary, local execution is more worthy[2].

2.2.7 Issues and Challenges in Computation Offloading

a. Reliability and Availability

Mobile computation offloading systems use wireless network for data transmission. There are several wireless technologies (WCDMA, WiMAX, GPRS, WLAN and CDMA2000) used by different mobile devices to access services provided by the cloud. The benefit and effectiveness of computation offloading is extremely dependent on the speed of the network, but the quality of wireless or cellular networks is highly unpredictable. Mobile computations may not be offloaded to the cloud due to network traffic congestion, lack of network coverage and various failures in the network. Enabling computation offloading in varying and unpredictable wireless network quality is an issue.

b. Low Bandwidth

Network bandwidth is the main issue of mobile cloud computing and computation offloading. Long delays caused by low bandwidth are a possible factor incurring network unreliability. During offloading the computation to the cloud server, the execution of the migrated task may suffer from long delays or even failures by the unreliable network [13]. The benefit gained from computation offloading (reducing execution time) can be highly affected by the unpredictable waiting time. A dynamic scheme to determine whether and when to launch the local re-execution, instead of always waiting for request timeout to offload [21] may solve this problem [13].

c. Security and Privacy

Many mobile applications use and store personal information related to banking, health, business, messaging, education and so on[14]. Due to the multitenancy of the cloud environment sensitive information sent to the remote server might be compromised and the privacy of the users might be violated. Traditional security protections (such as encryption, authentication, authorization etc.) are insufficient to secure computation offloading due to increased data transmission over networks with potentially unknown threats[48]. Developing modern security techniques that can secure the mobile data transmission, processing and storage in the cloud environment is an issue in computation offloading.

2.3 Summary

The rapid progress of mobile computing has transformed mobile applications into more complicated and computation intensive. However, mobile devices have limited resources (e.g., battery life, storage, and bandwidth etc.) and those limitations handicap the functionality of the mobile applications. As a solution to this problem, mobile cloud computing was introduced which is the integration of the resourceful cloud computing environment and the mobile computing technology. This brought new types of services and facilities for mobile users. The service includes the migration of computation intensive mobile application parts into the cloud environment for execution, which is known as computation offloading. Computation offloading is a promising technique to overcome the limitations of mobile devices.

CHAPTER THREE: RELATED WORK

3.1 Overview

Substantial number of researches have been done and several solutions have been proposed to improve the performance (i.e. shorten response or execution time) of mobile applications using different computation offloading techniques. Some of these proposed solutions also save energy at the client side simultaneously. The most prominent studies that are more related with the objectives of this thesis are discussed in this section.

3.2 Related Works on Optimizing Performance and Saving Energy Combined

Since enhancing performance and battery lifetime are crucial for mobile devices, a significant research efforts have been devoted to optimize those two objectives simultaneously[2].

3.2.1 CloneCloud

CloneCloud[7] is a computation offloading system that was aimed to improve the response time and energy consumption of mobile applications. CloneCloud utilized VM migration to offload a part of an application at the thread level granularity. The authors explained that application-layer virtual machines are widely applicable in mobile platforms (Dalvik VM). This helps to avoid having to modify executable applications and permits offloading these applications to more resourceful servers in the cloud. Internal state of the application-layer virtual machine was exploited to obtain the necessary information for executing the threads remotely. This state is also sent to the cloud and placed at the same address space as the virtual machine. This might simplify the whole offloading process but it consumes more bandwidth.

CloneCloud used the combination of static analysis and dynamic profiling to automatically partition the applications. The partitioning is made only once per application, the source code of the application is not required and the programmer doesn't have to write the application in a different way. Using the given set of execution conditions (CPU speeds, network characteristics and energy consumption) and by combining the static analysis with dynamic profiling, the application partitions are determined. This partitioning mechanism was said to improve the energy consumption and the performance of mobile applications. However, the partitioning is performed only once per application and a task will always be executed locally or remotely

according to the part where it is located without further consideration of the changes or fluctuations in the execution conditions.

CloneCloud proposed to continuously have a synchronized copy of virtual machine state to offload mobile applications. However, insufficient information is given about how the VM synchronization between the mobile device and the cloud is done, what performance and energy consequences it has. The authors of Cuckoo [25], argue that this technique is a costly technique that can be alleviated by temporarily offloading only the services that the mobile application requires during execution. CloneCloud selects the partition configuration from the database during run-time, which consider the availability of the cloud resources and network conditions. It also assumes stable network connectivity, which is unrealistic and the fluctuation of network quality is worth considering for better performance.

In conclusion, CloneCloud performs the whole VM state migration to execute the offloaded parts of an application. This increases the volume of data transfer, which degrades the performance of mobile applications especially in a network with low bandwidth.

3.2.2 MACS

MACS[6] is a computation offloading middleware that was proposed to improve the performance, energy and memory consumption of Android applications. Manual application partitioning mechanism was used in this system, which means the programmer separates the parts that should be executed remotely from the parts that should be executed locally. Information's such as the memory consumption of the application, the code size, the bandwidth and some dependency information are monitored and stored as meta data. This is used to calculate energy, time, and memory criteria that determine whether offloading would be worthwhile or not.

The consideration of the memory consumption of the applications for the offloading decision is one of the contribution of this offloading system. This feature of the computation offloading system is useful to offload applications that consume more memory than the available memory on the mobile device. The execution time and energy consumption of the application parts require an estimation, for making offloading decision based on these criteria. This estimation is done based only on the code size, which is insufficient to estimate the execution time and energy consumption of each application parts.

In conclusion, the manual application partitioning mechanism and dynamic offloading decision using only the code size is inadequate to identify computation and memory intensive parts of the applications. The execution conditions, intercommunication between each application parts and input parameters should be considered for a better result.

3.2.3 TDM

Other framework that shorten response time and reduce energy consumption of mobile applications simultaneously is proposed in[13]. Due to the divergence of response time and energy consumption a customizable cost function has designed, which allows end users to adjust the weight of response time and energy consumption and it's the particularity of this system. The system has two modules: factor measurement and ternary decision making. Before execution, static decision factors are measured, which are independent of a module and they are deterministic.

At a dynamic time, when a module is invoked, TDM first looks for the presence of the corresponding factor table of the application's module, which collects required parameters or device information to evaluate energy savings and execution time. If the factor table is not present, the module of the application is executed locally, and a creation of corresponding factor table process is completed. TDM insert static decision factors and module dependent decision factors to the factor table, which are provided by the module to the factor table. After completion of a module, factor table records execution time for future use. But, if the corresponding table of factors is present, TDM obtains the decision factors from the decision table. TDM then request cloud to give speed of execution and calculates network transmission bandwidth. By passing all the decision factors to the cost functions, the decision maker takes the offloading decision

Unlike the previous computation offloading systems, the offloading decision of TDM considers various decision parameter, which maximizes its accuracy. However, analyzing all of those decision parameters can affect the performance the application and increase the energy consumption of the mobile devices. Furthermore, the state transfer between the offloaded application parts and the mobile devices is not considered.

3.3 Related Works on Improving the Performance of Mobile Applications

Computation offloading is becoming an attractive solution for achieving performance requirements on mobile systems as applications are becoming increasingly complicated[2]. A

considerable number of researches have been done to optimize the performance of mobile applications using computation offloading.

3.3.1 COSMOS

In[10], the COSMOS framework is presented, which provides computation offloading as a service for mobile devices. COSMOS performs offloading at method level granularity to optimize the performance of mobile applications and to reduce the fee for the cloud services. The main aim of COSMOS was to solve the mismatch between how individual mobile device demand computing resources and how cloud providers offer them.

The paper addressed the fact that it takes a long time to lunch Cloud computing resources and are leased for a long-time span. Whereas offloading requests from a mobile device require quick response and may not be very frequent. The setup and leasing time of Amazon EC2 VM instance is taken as an example, which takes about 27 seconds to start and the shortest leasing time is one hour. Instead of provisioning resources for a constant time, COSMOS provides cloud resources based on the demand of the mobile devices, which is monitored by the COSMOS server instance.

On the other hand, the COSMOS Client component (or the application tracker) on each mobile device monitors application execution and network connectivity and makes offloading decisions. The application partitioning mechanism of COSMOS is the same as the partitioning techniques used in CloneCloud[7]. At the entry point of every compute-intensive task, the offloading controller obtains the computation speedup from the execution predictor and the communication delay from the connectivity predictor and decides if it should offload the task based on them. Based on the decision, the COSMOS Client allocates the worth offloading task to an active COSMOS Server. At the end, the COSMOS Client sets timeout and offloads the task. If the timeout expires before the result of the offloaded task is obtained, the COSMOS Client executes the task locally on the mobile device.

More specifically the following novel techniques has been developed, including:

- ✓ Resource-management mechanisms that select resources suitable for computation offloading
- ✓ Adaptively maintaining computing resources according to offloading requests

- ✓ Risk-control mechanisms that properly assess returns and risks in making offloading decisions
- ✓ Task-allocation algorithms that properly allocate offloading tasks to the cloud resources with limited control overhead.

In conclusion, COSMOS uses a manual process to profile the code of mobile applications. Since the runtime behavior of the application cannot be gained using such manual profiling, COSMOS hasn't considered the execution behavior of an applications during offloading decision. The paper explained COSMOS as system that provides computation offloading as a service to mobile devices but it's unclear how the offloading process is encapsulated as service oriented architecture. Moreover, no effort has been done to reduce data transfer between the mobile devices and the remote server during the execution of the applications.

3.3.2 Offload (only) the right jobs

A stochastic multi-queue model for dynamic computation offloading is proposed in[12]. It uses the various performance metrics (delay, energy, offloading costs) and the intermittently available access links (WLAN hotspots). The problem analysis is carried out in the Markov decision processes framework. The different options (process locally, or offload to a cloud, either using hotspot or via a cellular network) have been modelled as single server queues. The resulting dynamic decision problem has a specific type of task assignment problem, where the policy seeks to minimize a given cost function (e.g., the mean response time).

In this paper a near-optimal dynamic offloading policies have been derived that takes into account the different performance metrics (the availability of WLAN hotspots, energy consumption, communication costs and the expected delays), the state of the queues, and the anticipated future tasks when making the decisions. To evaluate the proposed policies a numerical experiment has been conducted. According to the results provided in the paper, the system can improve the performance (i.e. Minimized the execution time) of the mobile applications.

However, like the previous computation offloading systems the implementation of the proposed solutions can bring an additional overhead to the resource constrained mobile devices, due to the intensive analysis and computation requirement of the derived offloading policies.

3.3.3 To offload or not to offload

An efficient code partition algorithm for computation offloading is presented in[11]. The algorithm was designed to optimize the performance of computation-intensive applications. Instead of making an individual decision for each method like the previous computation offloading systems, the algorithm finds the offloading and integrating points for the whole application with all methods. The call graph model has been used to develop the algorithm. The proposed algorithm has Θ (V+E) time complexity, where V is the number of methods that can be offloaded and E is the number of invoking relations in the application.

The depth-first search (DFS) and a linear time searching scheme was used to find the offloading and integrating points on a sequence of calls. The algorithm considered both the execution cost and the communication cost of each methods in an application. According to this the methods that have longer execution time and that exchanges larger data during execution have a higher probability to be offloaded. This minimizes the execution state transmission between the mobile devices and the remote servers in the cloud environment, which improves the performance of the applications.

However, a timer was inserted in each method to determine the execution time in both the mobile device and the remote severs. Furthermore, the[49] tool was used to calculate the object's size in the heap and the result is considered as a communication cost. These techniques are computation intensive and they can affect the performance of the mobile applications. To alleviate these problems, the proposed system in this thesis performs application profiling and partitioning only at the server side. Moreover, the number calls between each method is considered as the communication cost instead of the size of the object.

3.4 Summary

The following Table 3.1 summarizes the previous computation offloading systems and compares these systems with our proposed system.

Table 3.1 The summary of related works

Papers	**Partitionin**	**Offloading**	**Achievement**	**Drawback**

	g method	decision		
CloneCloud[7]	Automatic	Dynamic	-Improve performance -Save energy	- Increases the volume of data(state) transfer during the execution of the applications -Assumes stable network connectivity
MACS[6]	Manual	Dynamic	-Improve performance -Save energy -Save memory	- Only the code size is used as decision parameter. -The state transfer between the mobile devices and the cloud during the execution of the applications wasn't considered.
TDM[13]	Automatic	Dynamic	-Improve performance -Save energy	-The offloading decision requires heavy computation which brings extra overhead to the mobile devices. -The intercommunication between the application parts is not considered
Papers	**Partitioning method**	**Offloading decision**	**Achievement**	**Drawback**
COSMOS[10]	Manual	Static	-Improve performance - Reduce the monetary	-Runtime conditions are not considered during

			cost of cloud services -Efficient resource allocation in the cloud environment	offloading decision -Increases state transfer between the mobile device and the cloud
Offload (only) the right jobs[12]	None	Dynamic	-Improve performance	-Introduces an additional overhead to the resource constrained mobile devices
To offload or not to offload[11]	Automatic	Dynamic	-Improve performance -The call relationship between each method is considered during partitioning	-Computation intensive techniques have been used to partition the applications -Application profiling and partitioning are performed at the mobile side.
Our proposed system	Both	Dynamic	-Improve performance -Reduce state transfer between the mobile devices and the cloud -Reduce the extra overhead incurred by application profiling and partitioning	-Previous execution time of the methods is used to partition the application -The offloading algorithm is application specific

CHAPTER FOUR: DESIGN OF THE PROPOSED COMPUTATION OFFLOADING SYSTEM

4.1 Overview

As its explained in the previous chapter the existing computation offloading systems are

sophisticated and can bring an additional overhead to the resource constrained mobile devices. To alleviate this problem a lightweight offloading system is proposed in this thesis. This system minimizes the execution time of mobile applications by using a lighter and simple offloading engine in the mobile side. The complex application profiling and partitioning that requires heavy computation are implemented in the server side. Moreover, instead of evaluating an individual method for offloading, the system determines the offloading and integrating points for the whole application with all methods. This chapter presents the design of the proposed computation offloading system and its components.

4.2 Characteristics of the Proposed System

4.2.1 Improve the Performance of Mobile Applications

The primary aim of the proposed system is to optimize the performance of computation-intensive mobile applications by reducing their execution time. Different techniques have been used to minimize the execution time of mobile applications. These techniques are;

- ✓ Considering the intercommunication (i.e. Call relationship) between each method during application partitioning
- ✓ Deploying the computation expensive components of the system at the server side
- ✓ Considering the quality of network during offloading decision
- ✓ Using asynchronous Remote Procedure Calls (RPC) for communication between the mobile devices and the remote servers in the cloud.

4.2.2 Client-Server Model

The VM migration and mobile agent communication models require a large state transfer during the execution of the applications[50], which affects the performance of the applications. Due to this reason, the conventional client-server communication model is used for this system. This model uses Remote Procedure Calls (RPC) or Remote Method Invocation(RMI) technologies to enable communication between the mobile devices and the cloud. Each of those communication technologies have their own pros and cons. Thus, a comparative analysis has been done in this thesis to choose the better between those technologies.

i. Comparison of RMI and RPC

Different computation offloading systems have used remote method invocation(RMI)[30][51], and remote procedure calls (RPC)[37][9] technologies for communication. Since those

communication technologies have different characteristics, it's important to determine which one of them is more suitable for computation offloading.

Remote Method Invocation(RMI)

Java RMI is a Java API that provides the ability to make calls from one JVM to another. RMI allows a developer to create distributed applications while retaining java compatibility and simplifying the overall complexity of a project. It provides the mechanism by which the server and the client communicate and pass information back and forth. Since the communication is performed through the network, RMI is dependent on the abilities to serialize object to turn an object into serial representation that is suitable for transmission and then reconstruct it on the receipt. This is necessary for remote methods that take objects as parameters as well as objects that have objects or return values[52].

Remote Procedure Calls (RPC)

Remote procedure call (RPC) technology is a standardized way of communication using a single protocol or a series of protocols, depending on the RPC implementation. One or more clients can connect to a server and exchange messages using RPC technology. Using RPC technologies enable every client on any operating system and platform to exchange messages with a server also running on any operating system and platform, as long as the RPC technology is supported (either as library or implemented in the application itself)[52].

The comparison of remote procedure call (RPC) and remote method invocation(RMI) is summarized in following Table 4.1.

Table 4.1 The comparison of RMI and RPC

Criteria	RMI	RPC
Programming paradigm	Object-oriented	Not object-oriented

Programming Language	It's a pure java implementation	It's programming language neutral
Invocation mechanism	Remote methods are invoked using proxy object	Remote functions are invoked using proxy function
Data size	During the remote method call the state of the object is also transferred within the method, which increases the size of data transfer.	Only the function is transmitted for remote processing, which requires less data transfer.
Performance	Slightly slower than the RPC because of the additional tasks involved to serialize the objects and because it must deal with the registry in order to communicate[52].	Since less computation is required to invoke remote functions, it is slightly better than RMI
Android support	Not supported by Dalvik VM[30]	Since RPC is programming language neutral, it is interoperable with different programming languages and platforms.

Inconclusion, we chose the Remote Procedure Calls (RPC) technology for the proposed system. As its explained in the above table, RPC is better than RMI in performance which is useful to achieve the main objective of this thesis. Moreover, when a remote method is invoked using RMI the state of the object is also transmitted within the method. This increases the data transfer between the offloaded application parts and the ones that are executing in the mobile device. Due to this reasons RPC-based approach is well suited for computation offloading systems. Furthermore, the asynchronous type of RPC is used to enable the execution of the remaining application parts at the mobile device in the meantime.

4.2.3 Method-Level Granularity

The proposed system offloads the mobile applications at the method level granularity. This is because it is the best granularity to analyze the computation requirement and to obtain the different parameters for offloading decision. Since the remote procedure call (RPC) enables the execution of functions on a remote system, this is the only granularity which is compatible with this technology. To obtain those benefits the proposed computation offloading system partitions the application at a method level and offloads the methods that are worth offloading.

4.2.4 Automatic and Manual Partitioning

First of all, the proposed system allows the developers to manually identify some of the methods that should be executed in the mobile device. Mainly those are the methods that directly interact with input/output devices or/and with the interfaces of the mobile application. In the first execution of the application, the proposed system offloads all of the methods in the application except the ones that are identified by the programmers for local execution. Then an automatic partitioning is performed using the call graph model in the cloud environment forthcoming execution of the application. The critical path method is used for partitioning the applications into several groups and the Java annotation is used to differentiate those groups (partitions) and their priority.

4.2.5 Dynamic Decision Making

The offloading engine of the proposed system offloads every method that are specified by the developers for remote execution. Then after the partitioning algorithm in the cloud environment partitioned the application and prioritized those partitions, the offloading engine in the mobile device processes the annotation of each partition and offloads those partitions based on their priority. For the later scenario, the quality of the network is also considered for decision making.

4.2.6 Application Partitioning and Profiling in the Cloud Environment

Application partitioning and profiling are the common components of computation offloading systems that require heavy computation. Implementing those components in the client-side introduces an extra overhead to the resource constrained mobile devices, which can maximize the execution time of the applications. To solve this problem those components of the proposed system are deployed in the cloud environment.

4.3 System Architecture

The architecture of the proposed system is depicted in the following Figure 4.1. The proposed system architecture consists of four layers, several components and subcomponents in each layer. Each of those components and subcomponents perform specific tasks and have different functionalities.

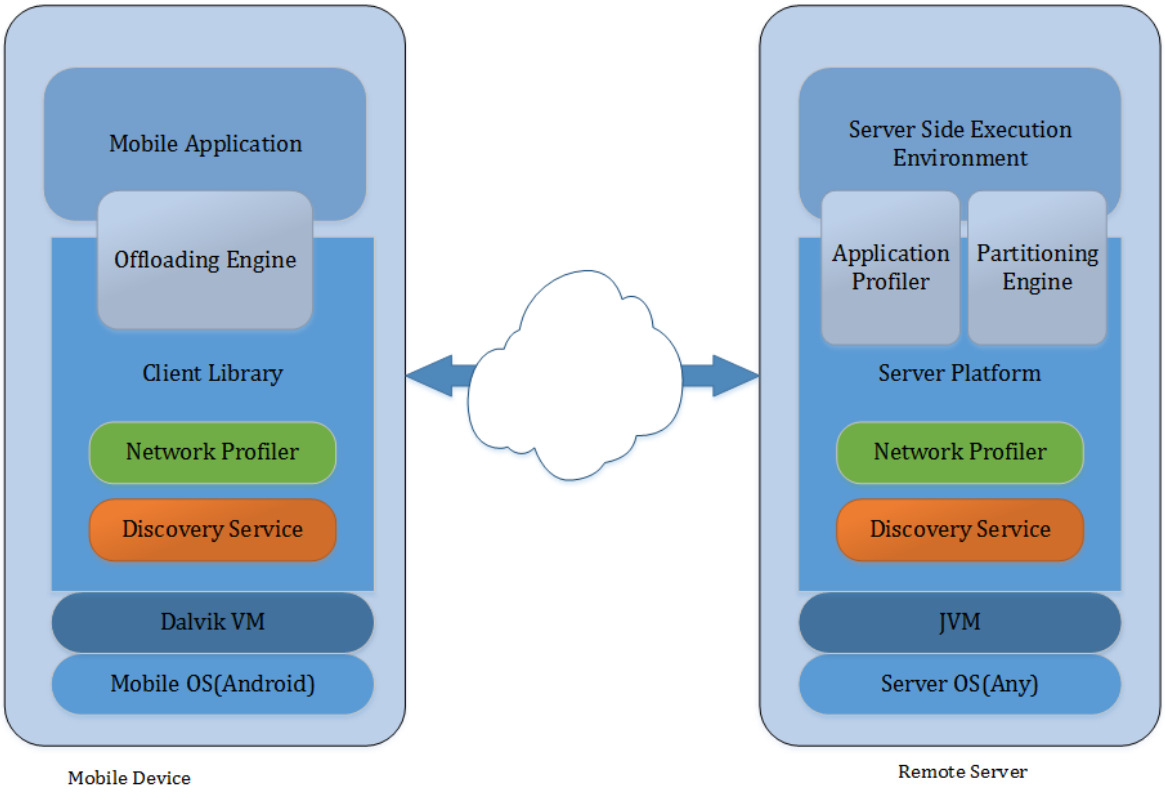

Figure 4.1 The architecture of the proposed system

The first layer of the system architecture has the operating system of both the mobile device (Android) and the remote server (any OS). The second layer includes the Java Virtual Machine(JVM) for remote servers and the Dalvik Virtual Machine(DVM) for Android operating system. The DVM is designed for the low memory environments and running multiple virtual instances on Android at the same time[44]. Like the Java bytecode (.class) can run on the JVM, Dalvik bytecode (.dex) is a special kind of code which could only be run on DVM. Due to this reason the remote server cannot run the mobile application directly, it requires translation of the

Dalvik bytecodes into the Java bytecodes that can run on the JVM. The interoperability of RPC with different programming language and platform provides a conducive environment to alleviate this issue.

The third layer consists of the Client Library in the client side and the Server Platform in the server side. The Client Library contains the Network Profiler, Discovery Service and the Offloading Engine components. The components of the Server Platform are Discovery Service, Network Profiler, Application Profiler and Partitioning Engine. Network Profiler and Discovery Service are the components that works in both the client and the server side. The service discovery technique in MpOS[53] framework is used with some minor modifications to integrate it with the other components. Furthermore, the Network Profiler is developed by combining some of the features implemented in MpOS[53] framework and in[54]. The code of several call graph projects in[55] were also used as a base to develop the Application Profiler.

The fourth top layer is represented by Mobile Application on client side and Server Side Execution Environment (SSEE) on server side. Server Side Execution Environment (SSEE) are set of server instances that are responsible for executing the offloaded parts of the applications.

4.3.1 Client Side Components of the Proposed System

i. Discovery Service

This component of the system is responsible for discovering a running server platform on the same wireless LAN and initializing a communication session with them. Whenever the mobile application is launched, the mobile device announces its presence by sending a multicast message to the WLAN. The Server Platform has a service listening for multicast messages in the WLAN. When the multicast message arrives, the Server Platform sends a message back to the mobile device for informing the services that are available for the application. Using this information, the mobile application interacts with the remote server and obtains the right service. The component is taken from MpOS[53] framework with some minor modification for integration with the other components.

ii. Network Profiler

The network profiler is a component of the proposed system that monitors and measures the quality of the network connection between the mobile device and the remote server. The round-trip time (RTT) is used as a metrics to measure the strength of the network. A ping request is sent

to the remote server and the round-trip time (RTT) is measured. This operation is performed every 45 seconds and the result is updated after each operation. The result is used by the offloading engine as a parameter for identifying the methods that should be offloaded and that shouldn't.

iii. Offloading Engine

The offloading engine performs offloading decision based on the result obtained from the Network Profiler and the partitioning performed in the cloud environment. At the first execution of the application the offloading engine offloads every method that are manually identified by the developers for remote processing. Java annotation has been used to identify those methods and in our case ***@Remotable*** annotation is given to the methods that should be offloaded to the cloud environment. This helps to move the application profiling and partitioning processes to the remote server because those processes require heavy computation.

In our proposed system, we assumed that the processing capability of the remote server is always better than that of the mobile devices. Based on this assumption performing the application profiling and partitioning at server side makes the offloading decision lighter and improves the performance of the mobile applications. After the first execution of the methods in the remote server, the offloading engine begins to consider the partitioning performed by the Partitioning Engine in the remote server. In this thesis, a lighter offloading algorithm which is depicted in the following Figure 4.2 is designed for offloading decision.

Using Round-Trip Time for Offloading Decision

The round-trip time is defined as the elapsed time between the instant a packet is sent by the source node to the instant the corresponding ACK packet is received by the source node[56]. Round-trip time (RTT) can be used to analyze and estimate the speed of a network connection. We used the round-trip time (RTT) in offloading decision because it's simple for analysis and requires lower computation, which reduce overhead on the mobile device.

Determining RTT boundary

Computation offloading systems that use RTT as parameter for offloading decision have different RTT boundaries for performing offloading. In[53] computation offloading is only

performed when RTT is lower than 40ms (40 millisecond). To determine the RTT boundary for the proposed system, an experiment was conducted. In the experiment, the proposed system was observed in different RTT values. A static offloading decision was configured for performing the whole and partial offloading of the mobile application in varies RTT values ranging between 10ms to 100ms. Based on the result, computation offloading was worthwhile when RTT is lower than ≈32ms. This RTT value is used as a boundary in the offloading decision, which means the proposed system performs offloading when RTT is lower than 32ms.

```
1.  // INPUT: Method, RTT, Method_Annotation, Partitions, Partition_Priority
2.  // OUTPUT: Performs offloading if its worthy else executes the method locally
3.  Intercept method
4.  IF Method_Annotation == Remotable THEN  //Offloads all method in the first execution
5.      Offload the method to remote server
6.  ELSE IF RTT ≤ 32 THEN
7.      Determine RTT difference (RTTD = 32 – RTT)
8.      Sort the Partitions ascending depending on Partition_Priority
9.          FOR each Partition DO
10.             IF Partition_Priority ≤ RTTD THEN
11.                 Offload the Partition to the remote server
12.             END IF
13.         END FOR
14. Else
15.     Execute the method in the mobile device
16. End if
```

Figure 4.2 Pseudo code of the computation offloading algorithm

4.3.2 Server Side Components of the Proposed System

i. Application Profiler

In the existing computation offloading systems, in order to decide whether it is worth it or not to offload a given method, in a given situation, the system should determine or estimate the execution time for both the local and the remote execution. The lowest execution time determines where the method will be executed. Profiling the application is the common and essential mechanism to determine or estimate the execution time of each method. Our

Application Profiler is implemented only in the server side to reduce overhead in the mobile device side. The java instrumentation API is used to develop the Application Profiler and the Partitioning Engine.

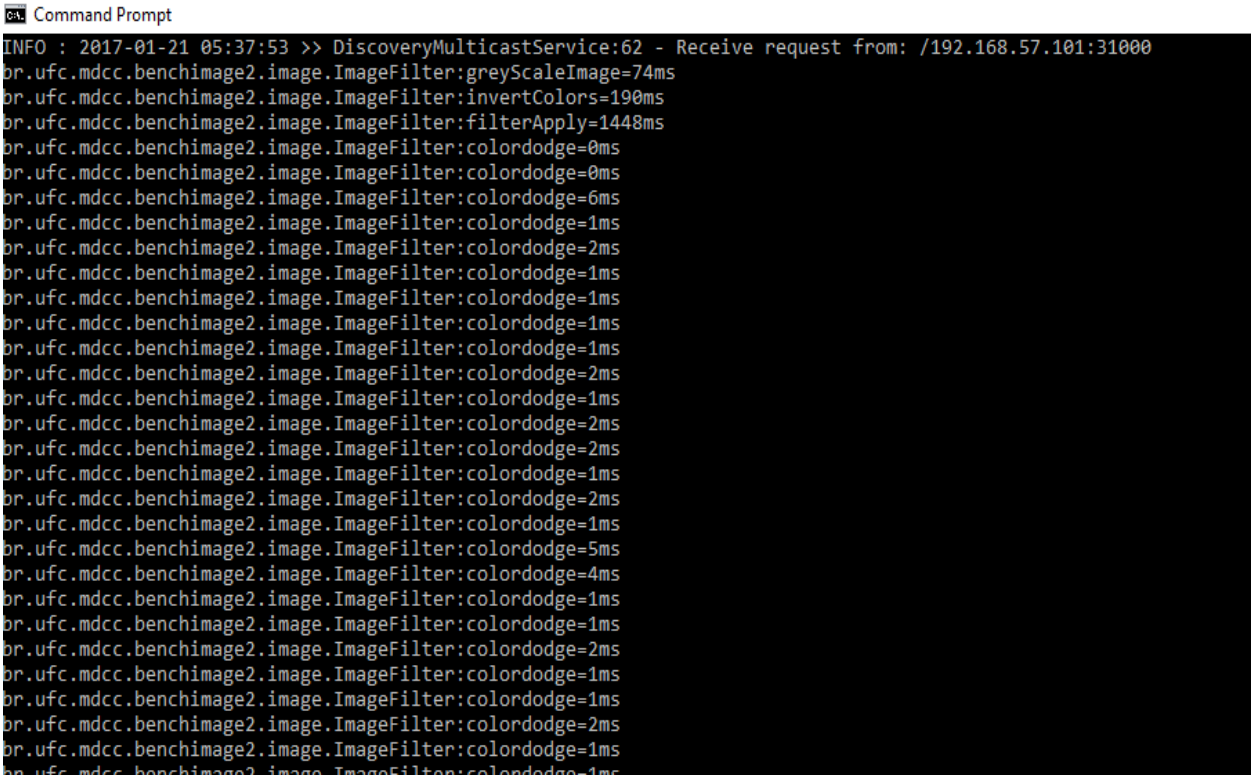

Figure 4.3 Screenshot of the Application Profiler

Java Instrumentation

The java instrumentation helps to access a class that is loaded by the Java ClassLoader from the JVM and modify its bytecode at runtime. Mostly profilers, application monitoring agents, debuggers, event loggers use Java instrumentation. Java instrumentation contains four key components[57]:

- Agent: is a jar file containing agent and transformer class files.
- Agent Class: a java class file, containing a method named ‘premain’.
- Manifest: manifest.mf file containing the “premain-class” property.
- Transformer: a Java class file implementing the interface ClassFileTransformer.

Our Application Profiler includes those components and the Javassist instrumentation library had been used to simplify the development of the Application Profiler. Javassist is an open source class library for manipulating Java bytecode[58]. The Application Profiler generates a call graph by determining the execution cost of each method and the number of calls (i.e. the communication cost) between each method during execution, which is used later by the Partitioning Engine to partition the application.

Call Graph Model

The call graph is a directed weight graph where the vertex represents a method and the edge represents the calling relationship between the methods[11]. In our case the vertex represents the execution time of the method (i.e. Execution cost) and the edge of the graph represents the number of calls between each method (i.e. Communication cost). Figure 4.4 shows the call graph of cartooning an image using the BenchImage application, which was used to evaluate the proposed computation offloading system. The methods are labeled with letters, the numbers inside the circle represents the execution time of the methods in millisecond and the numbers in the edges represent the number of calls between the methods.

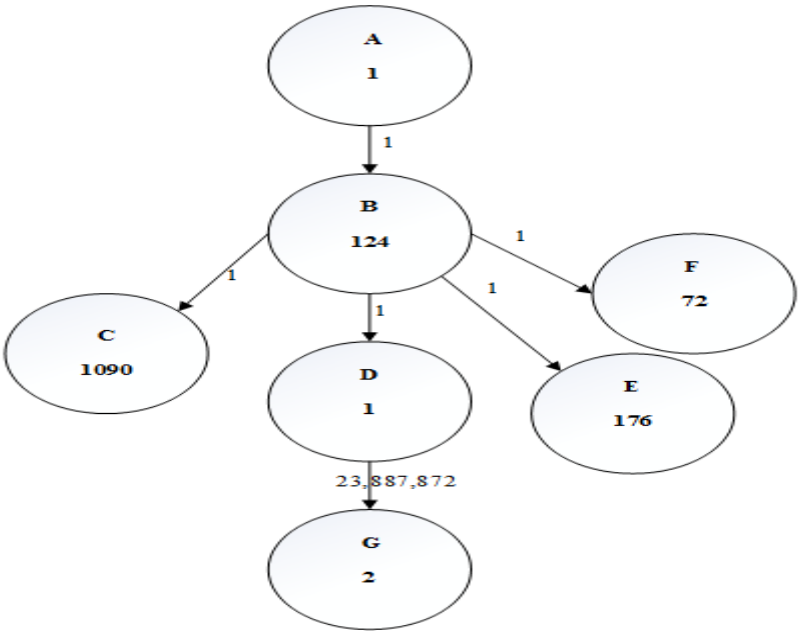

Figure 4.4 Call graph of cartooning an image using BenchImage

ii. Partitioning Engine

The call graph generated by the Application Profiler is taken as an input by the Partitioning Engine. It performs partitioning by using the critical path method, which determines the longest path through the call graph. This technique was used in[59] for scheduling application tasks in heterogeneous systems which have processors with different characteristics. A significant modification has been done to adopt this scheduling algorithm. The proposed application partitioning algorithm partitions the application and prioritizes the offloaded methods for future execution. It uses Java annotation to differentiate and prioritize the partitions, and these annotations are updated in each partitioning process by the partitioning algorithm at runtime.

Application Partitioning Algorithm

We modelled all offloaded methods as a set of vertices V, and all their call relationships as a set of edges E, using the results provided by the Application Profiler. Those forms a directed weight graph G = (V, E), namely the Call Graph. The partitioning algorithm takes this call graph as an input to partition the applications. The weight of the vertices depicts the execution cost (i.e. execution time) of each method in the cloud. The edges of the graph are labeled with the number of calls made between each method during execution.

Unlike the other computation offloading frameworks, we only considered the execution cost of the methods in the remote server. In this thesis, we argue that profiling the execution of the application in the mobile device brings additional overhead to the mobile devices. We assumed that the execution of a method (or an application) in a mobile device always takes longer time than the execution of a method (or an application) in a server. The only way the execution time in the cloud can be longer is due to the delay in the network connection. To fill this gap, the proposed system considers the quality of the network connection between the mobile device and the remote servers in offloading decision.

first, a start node and an exit node to represent the beginning and end of the applications execution respectively are added to the call graph. Pseudo-edges are also added to the call graph between the methods that doesn’t call other methods but are not the last methods to be executed and the exit node, which helps to maintain the integration of the call graph. The partitioning algorithm uses the critical path method to partition the mobile applications. A critical path is the longest path from the start node to the exit node in the call graph. The length of a path is

computed as the sum of the vertices weight (i.e. the execution costs of the methods) and the weight of the edges (i.e. number of calls) along that path.

Obtaining a single critical path might not be enough for an application with large number of methods and complicated call graph. The remaining methods in the call graph may still be computation intensive and requires further partitioning. To solve this problem, our algorithm continues to determine the critical path in the remaining call graph after pruning the obtained critical path. Besides this the algorithm changes the annotation of the methods to a common annotation through the partition (i.e. critical path) and unique for each partition. According to the partitioning phase the partitions have a priority which is used by the offloading algorithm for decision making. Which means the partition (i.e. the methods in the critical path) acquired in the first partitioning phase has a higher offloading priority than the partition acquired in the second partitioning phase and so on. The following figures illustrate how the algorithm partitions an application.

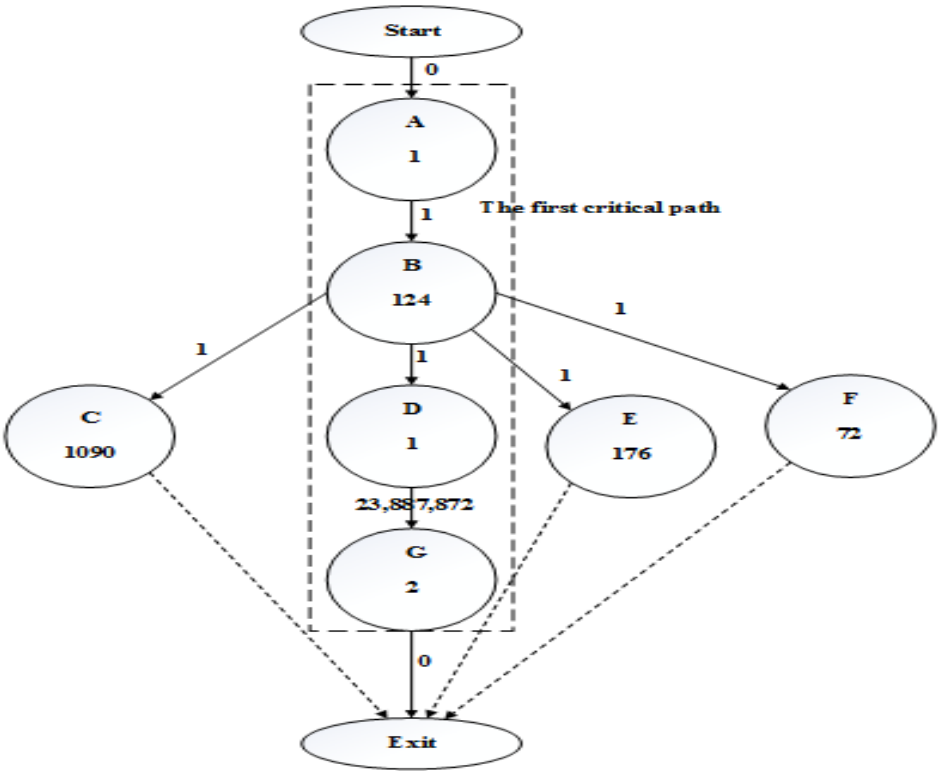

Figure 4.5 The first critical path in the call graph of the BenchImage application

The path through ABDG is the first critical path (i.e. the longest path) in the above call graph.

The algorithm changes the annotation of the methods in the critical path and prunes those methods from the call graph.

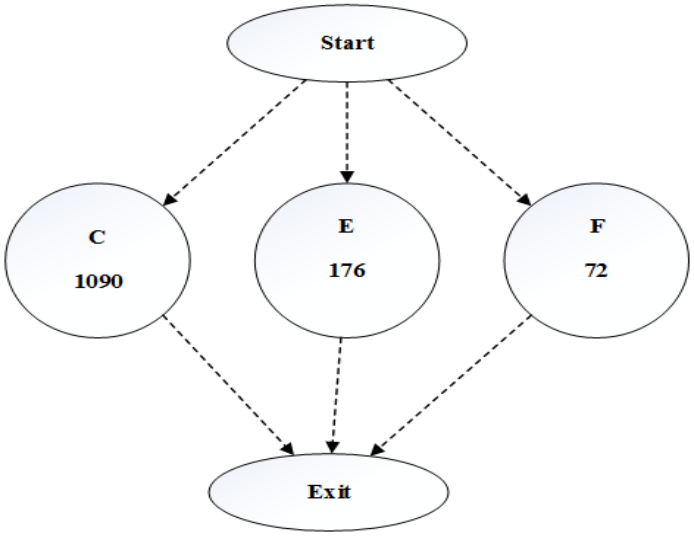

Figure 4.6 The call graph after pruning the first critical path

Since the remaining nodes in the call graph are single and isolated the partitioning algorithm sorts those method nodes using their execution cost and changes the annotation of those methods.

Table 4.1 The terms that are used in the application partitioning algorithm

Terms	Definition
CP	Critical path
Pred(u_k)	The method that calls u_k method (i.e. predecessor of node u_k)
Succ(u_k)	The method that is called by the u_k method (i.e. successors of node uk)
E_{CP}	The edges of the nodes in the critical path
G	Call graph
V_T	Set vertices (nodes) in the call graph
E	Edges of the call graph

```
1.  //INPUT:  G = (V_T, E); the call graph of the application
2.  //OUTPUT: Optimal partitions of the call graph
3.  count = 1
4.  while there are method nodes in the call graph do
5.  Initialize Length (v_i) = 0, for each v_i ϵ V_T        // Find the length of the critical path
6.    for each v_i ϵ V_T do
7.      for each edge v_i → w_j do
8.        if Length(w_j) ≤ Length(v_i) + Execution Cost(w_j) + Edge Weight(v_i → w_j) then
9.          Length(w_j) = Length(v_i) + Execution Cost(w_j) + Edge Weight(v_i → w_j)
10.         PredCost(w_j) = v_i
11.       end if
12.     end for
13.   end for
14. loc = 1, maxL = 0
15. for k = 1 to |V_T| do                 // To find the last node of the critical path
16.   if Length(v_k) ≥ maxL then
17.   maxL = L(v_k)
18.    loc = k
19.   end if
20. end for
21.   node = V_loc                        // Let node be the last method of the critical path
22. CP_count = {V_loc}
23. while node ≠ start do                 // Add nodes to the current critical path CP_count
24.   node = PredCost(node)
25.   CP_count = CP_count U node
26. end while
27. G'= (CP_count, E_CP)                  // E_CP are the edges of the nodes in CP_count
28. G = G - G'                            // Prune the nodes in the critical path from the call graph
29. for each u_k ϵ CP_count do
30.    Annotation(u_k) = count            // Change the annotation of the methods
31.   if |Pred(u_k)| == 0 then            // Add pseudo-edges to the call graph
32.    G = G U (null, start → u_k)
33.   else if |Succ(u_k)| == 0 then
34.    G = G U (null, u_k → exit)
35.   end if
36. end for
37. count+ = 1
38. end while
39. return CP
```

Figure 4.7 The application partitioning algorithm adopted from[59]

4.4 Execution Flow of the Proposed System

The proposed computation offloading system starts by intercepting the local execution of the methods and examining the annotation of those methods. If the annotation of the method is ***@Remotable***, it will understand that this is the first execution of the method and the method will be immediately offloaded to the remote server. During the execution of the methods in the remote server the system performs partitioning and modifies the annotation of the methods. The

annotation of the methods defines their partition and the priority of the partitions for computation offloading. The offloading decision is made using the priority of the partitions and the quality of the network. The detailed description of how the proposed system works during the execution of a mobile application is presented in the following Figure 4.8.

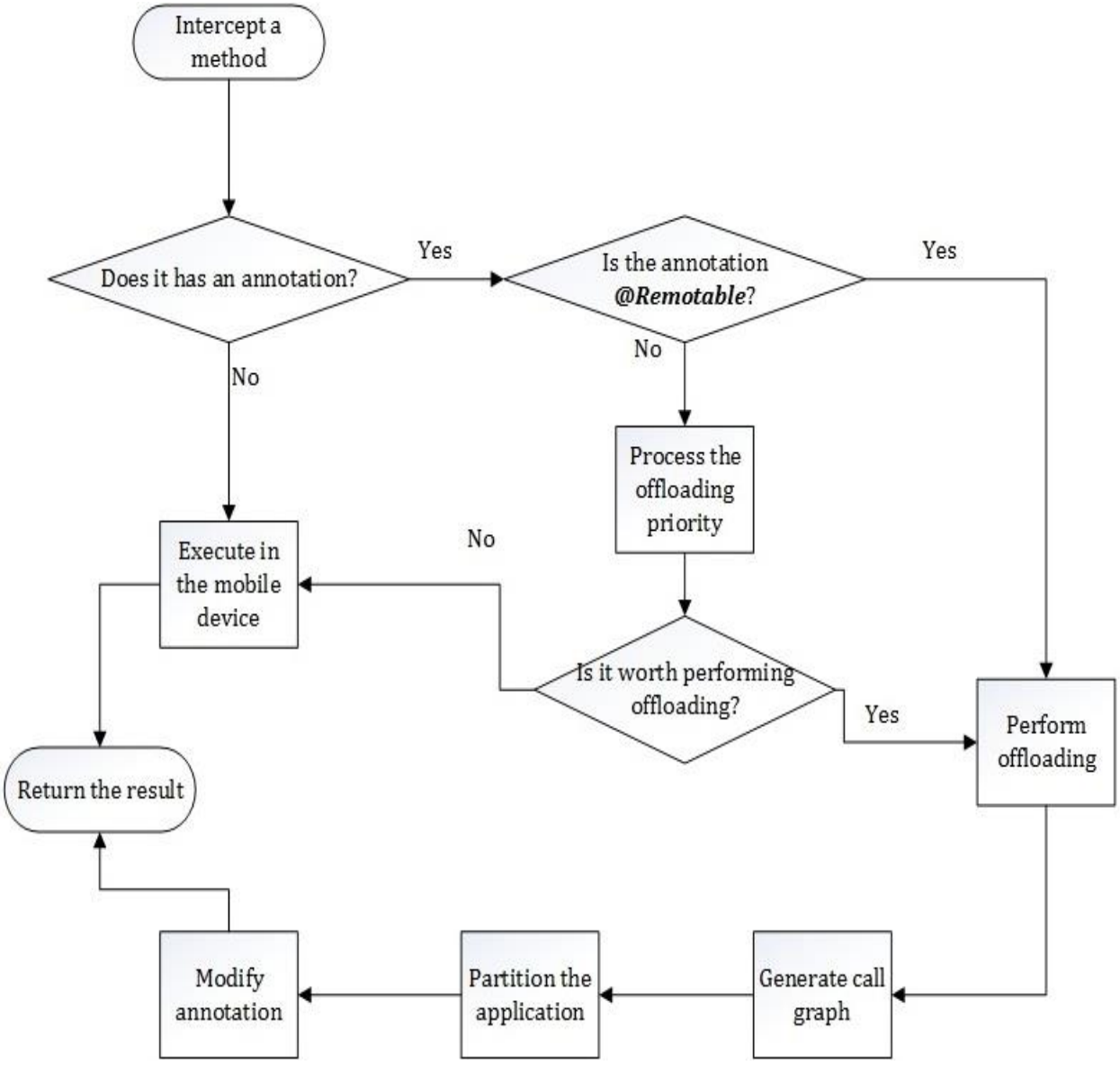

Figure 4.8 The execution flow of the proposed system

4.5 Summary

The proposed computation offloading system performs offloading at method level granularity to optimize the performance of computation-intensive mobile applications. To achieve this, several different techniques were considered and incorporated to the proposed system. The proposed computation offloading system uses the critical path method to partition and reduce the total execution time of the mobile applications. It also uses the RPC technology for communication between the mobile device and the remote server during offloading operation. Besides those the computation intensive components of the proposed system were implemented in the server side. This reduces overhead in the mobile device side and improves the performance of mobile applications.

CHAPTER FIVE: IMPLEMENTATION AND EVALUATION

4.1 Implementation of the Proposed System

The proposed computation offloading system is built only for Android mobile operating system. Some configurations are required to enable the mobile applications to use this computation offloading system. The developers need to perform the following configuration to integrate the proposed system with mobile applications.

- Developers must annotate the methods that are candidates for the offloading operation with the ***@Remotable*** annotation. This annotation must be made using interfaces so the concrete classes that implement interfaces can support offloading. The use of interfaces is required because they enable the system to create proxy of objects, which allows to access the methods[53]. This enables the interception of the methods from the execution flow of the application and perform offloading operation.
- The Client Library must be referenced as a library by the mobile application project.
- To run the Server Platform, the developers must manually configure the IP address of the cloudlet in the config file. The config file, jar file of the Server Platform and jar file of the application profiler and partitioning should be in the same folder.

4.1.1 Testbed Setup

The cloudlet computation offloading architecture is used to implement and evaluate the proposed computation offloading system. A laptop computer has configured as a cloudlet (i.e. the remote server) and an android mobile device is used to test the system. Those two devices have been connected to the same WLAN (wireless LAN).

Hardware Specifications

The devices used to test the proposed system and their technical details is presented as follows:

1. Mobile device
   - ✓ Mobile device: Asus
   - ✓ Android version: 5.1
   - ✓ Processor: ARMv7 Processor rev3(v71)
   - ✓ Main memory: 2 GB

2. Computer
   - ✓ Processor: 2 cores, Intel(R) Core(TM) i5 CPU M560 2.67 GHz.
   - ✓ Main memory:4 GB.
   - ✓ Java version: 1.8
3. Wireless device
   - ✓ Vendor: TP-Link Technologies
   - ✓ Link speed: 135Mbps

Software Specifications

The development environment was composed of the following different software's.

- ✓ Eclipse Neon
- ✓ Android SDK (Software Development Kit)
- ✓ ADT Plugin for Eclipse (Android Development Tool)
- ✓ Connectify Hotspot
- ✓ Maven

The developing framework uses Eclipse, the Android SDK, the ADT plug-in, the Connectify Hotspot and Maven. The Android SDK is a package that provides API libraries and development tools necessary to build, test and debug apps for Android. The ADT plug-in for Eclipse allows setting up Android projects, creating an application UI, adding packages based on the Android Framework API, and provides an emulator to test the Android applications in the development machine[60]. Since wireless routers are expensive, the Connectify Hotspot software was used to connect the mobile device and the cloudlet during the development of the system, which turns a computer with wireless adapter into a wireless router. Maven was used to manage the different components of the proposed system and their dependency.

4.2 Evaluation of the Proposed System

The proposed computation offloading system was evaluated to assure that its objectives were achieved. Since improving the performance of computation-intensive mobile applications is the main objective of this research, the performance of the proposed system was evaluated under different execution conditions. To examine the performance of the proposed system, a computation-intensive mobile application called BenchImage has been used.

This mobile application was chosen because it was specifically built for testing the performance of computation offloading systems. It has a built-in profiler which measures the execution time of every computation performed by the application. This minimizes the effort required to integrate the proposed system and the mobile application with other performance profiling tools. It also has pre-stored images with different resolutions, which can be used to evaluate the performance of the proposed system at different input data size (i.e. resolution).

4.2.1 The BenchImage Application

BenchImage is an open source Android application developed in MPOS framework[53] and used to test the framework. It is a mobile application that uses image processing algorithms to apply some effects (i.e., filters) on images. Image editing effects such as cartooning, sharpening and applying Red Ton to the images are included in the mobile application. The application has several pre-stored images with different resolutions ranging from 0.3 MP to 8 MP. The user can choose one of these stored images and apply desired effect on them[53].

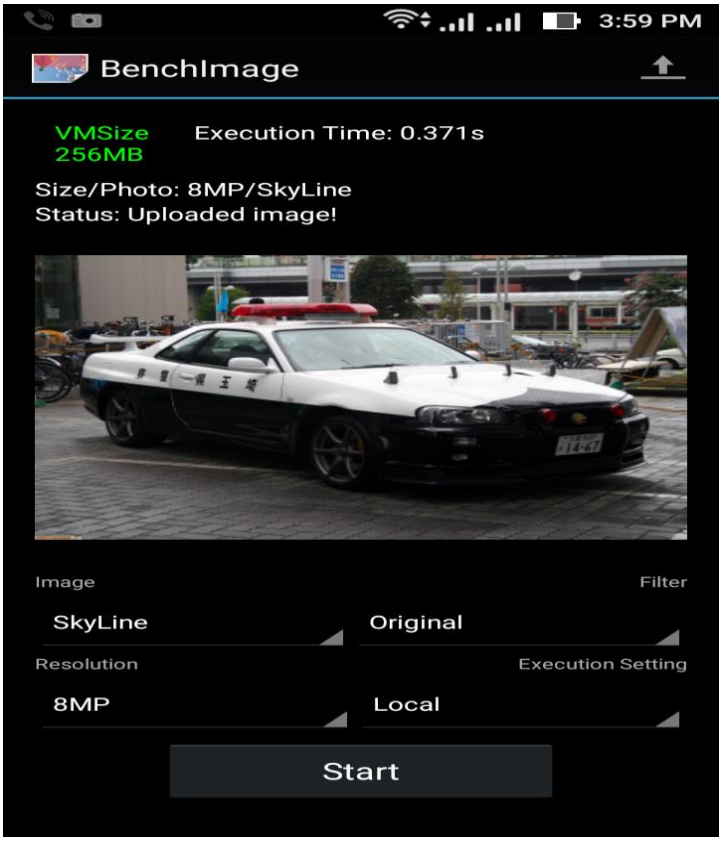

Figure 5.1 Screenshot of the BenchImage application

The BenchImage application has four main menus:

- ✓ Image: allows user to select one image from the available images
- ✓ Resolution: used to specify the resolution of the selected image
- ✓ The execution environment: allows users to choose the execution environment (i.e. locally in the mobile device or offload to the remote server).
- ✓ Effect (filter): contains the effects to be applied to the image

The Effects (Filters)

The BenchImage application has several effects that can be applied to an image, including:

Sharpening

Sharpening is a technique for increasing the apparent sharpness or smoothening of an image. There are three main reasons to sharpen an image, which are to overcome blurring introduced by camera equipment, to draw attention to certain areas and to increase legibility[61]. Sharpening is one of the effects that are included in the BenchImage mobile application.

Applying Red Ton

This effect is used to apply a red color to the images. Applying different weighting of the colors will result in totally different images. This effect is simple a color conversion, which is effusing visual shocks by the means of changing color. For example, when converting gray scale of a photo, usually the result of the processing is a Black and White image. The same is applying blue hue and in return, the result is a blue color image. So, does making a red color image[62].

Cartooning

Applying the cartooning effect to an image is the most computationally intensive process in the BenchImage application. Several other effects are applied in sequence to cartoon an image. The sequence of these effects is as follows[53]:

1. Transformation of the original image to grayscale
2. Cloning the image and reversing its colors
3. Execution of the Sharpen filter to blur the image generated in (1)
4. Execution of the “color dodge blend” operation, which blends the image generated in (2) and (3).

As a result, a new image that looks like a pencil sketch is created and shown to the user in the application display. Besides this the cartooned image is stored in the BenchImageOutput folder in the mobile device.

4.3 Results and Discussion

The performance of the proposed computation offloading system was measured using the execution time (response time) of the mobile application. The execution time of the mobile application in the mobile device and in offloading to a remote cloudlet was measured. Those execution times were compared to determine how much offloading using the proposed system improved the performance (or minimized the execution time) of the mobile application. During the evaluation of the proposed system the upload and download speed of the network was varying between 1.08Mbps and 3.20Mbps.

To evaluate the performance of the proposed computation offloading system, the BenchImage application was observed in applying three different effects to an image. These effects are;

4.3.1 Applying Cartooning Effect

Execution time of the application in applying cartooning effect was examined to evaluate the performance of the proposed system. An image with different resolutions (0.3MP - 8MP) were used to determine this execution time. To obtain a more accurate result the cartooning effect was applied 100 times to each image resolution and an average execution time of those executions was computed. The following table 5.1 shows the average execution time of cartooning an image locally in the mobile device and in offloading to the remote cloudlet.

Figure 5.2 Screenshot for cartooning an image in the mobile device

Figure 5.3 Screenshot for cartooning an image in offloading

Table 5.1 Average Execution Time for Applying Cartooning Effect (seconds)

Execution Setting	**Image Resolution (MP)**				
	0.3	1	2	4	8
Local	1.906	6.638	12.509	24.145	49.771
Offloading	0.337	1.444	2.616	4.481	8.788
Performance Improvement	5.65X	4.59X	4.78X	5.38X	5.66X

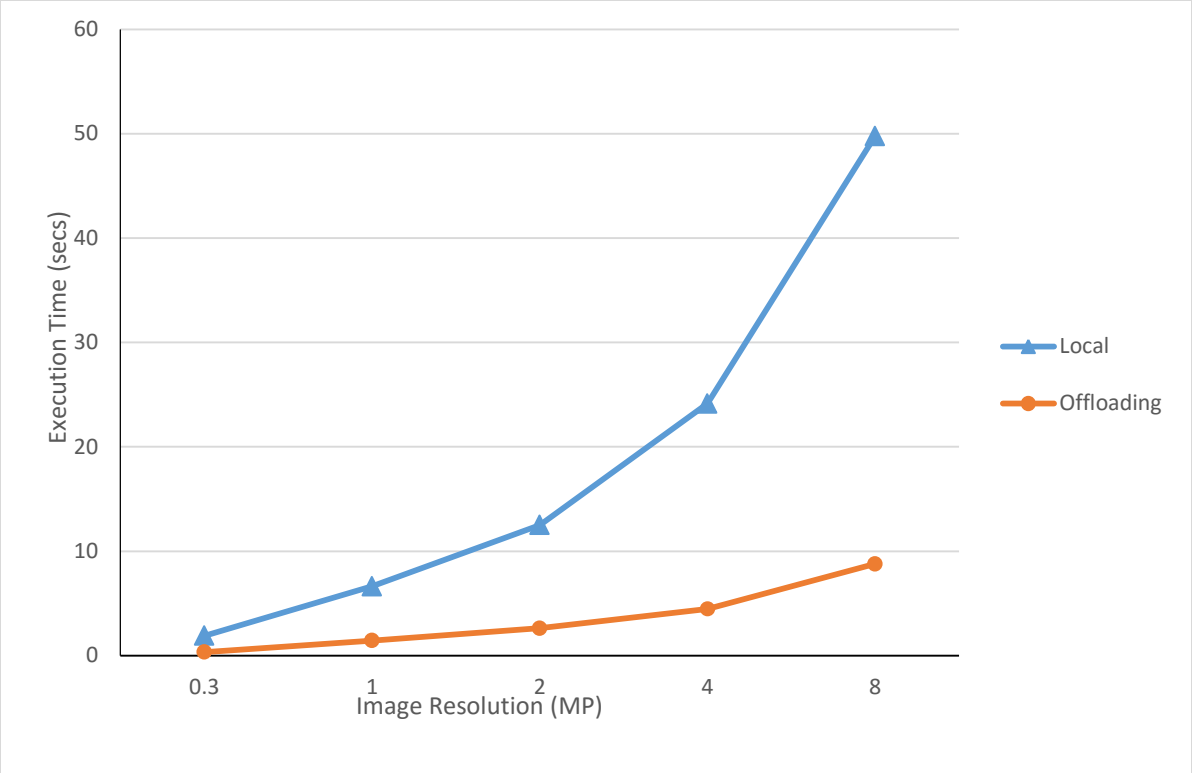

Figure 5.4 The result of cartooning an image

4.3.2 Applying Sharpening Effect

Sharpening an image was also used to evaluate the performance of the proposed computation offloading system. The same evaluation procedure as the cartooning effect have been followed to determine the average execution time of applying sharpening effect to an image. The following table 5.2 shows the average execution time for applying sharpening effect to an image.

Figure 5.5 Screenshot for sharpening an image in the mobile device

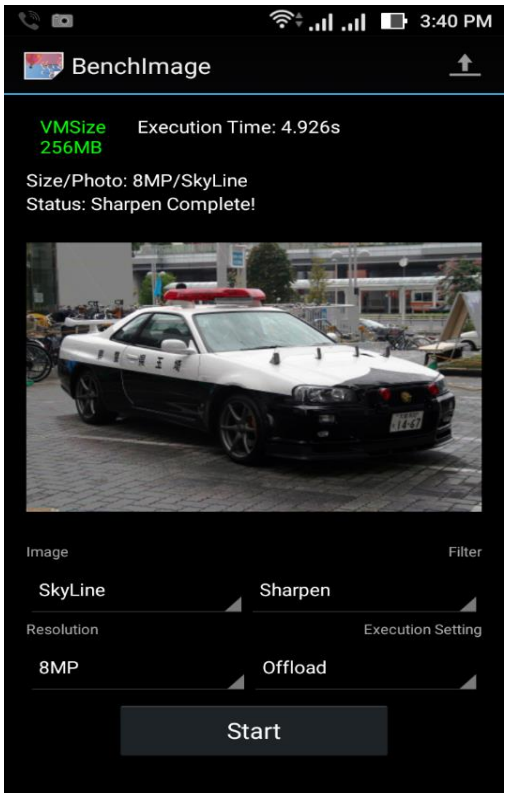

Figure 5.6 Screenshot for sharpening an image in offloading

Table 5.2 Average Execution Time for Applying Sharpen Effect (seconds)

Execution Setting	**Image Resolution (MP)**				
	0.3	1	2	4	8
Local	1.341	4.665	8.563	16.623	34.812
Offloading	0.508	0.640	1.192	1.901	4.710
Performance Improvement	2.63X	7.28X	7.18X	8.74X	7.39X

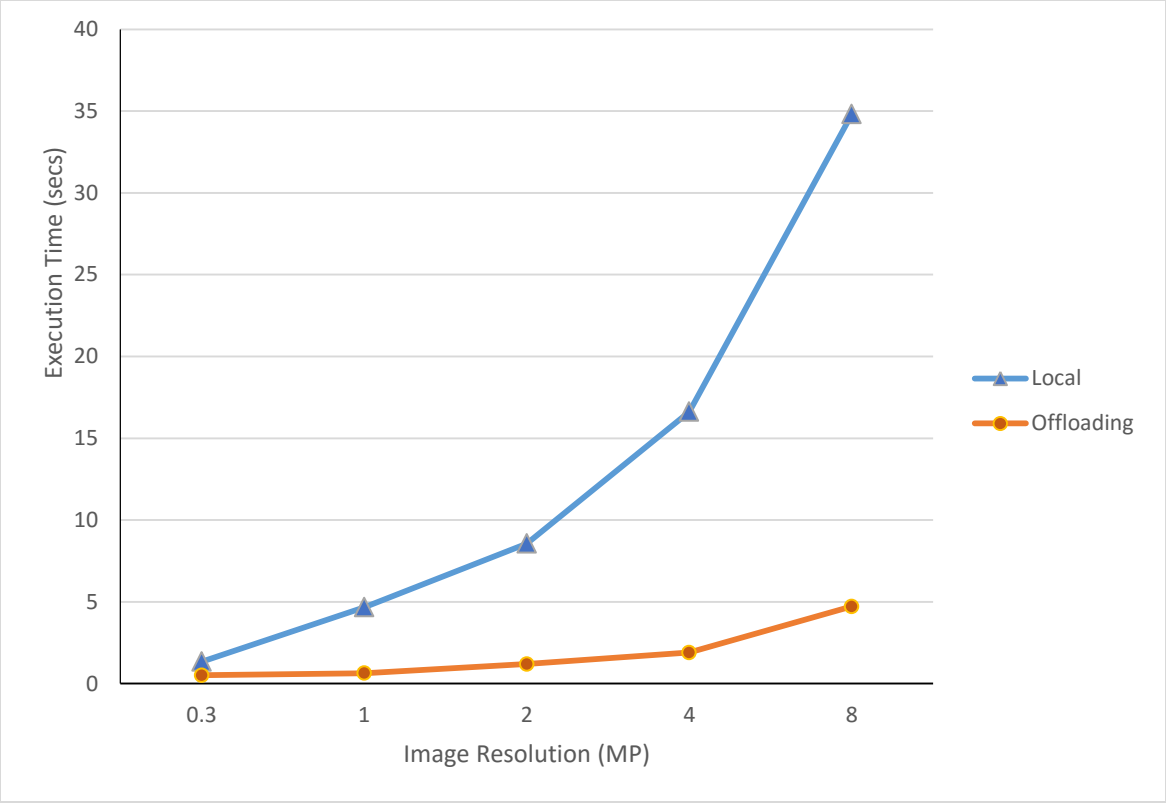

Figure 5.7 The result of sharpening an image

4.3.3 Applying Red Ton Effect

The third effect that was used to evaluate the performance of the proposed computation offloading system is the Red Ton effect. The average execution time of applying this effect to an image with different resolutions have been used to compare the application performance in the local mobile device and in offloading to the remote cloudlet. The following table 5.3 shows the average execution time for applying Red Ton effect to an image.

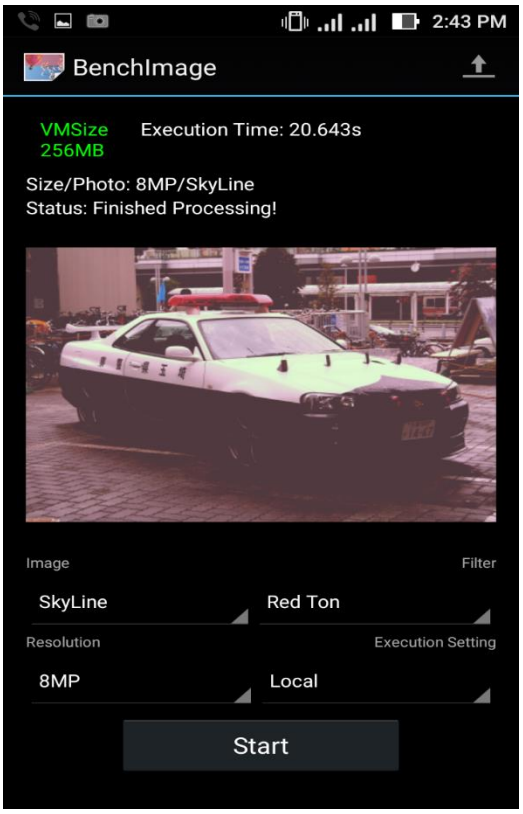

Figure 5.8 Screenshot for applying Red Ton effect in the mobile device

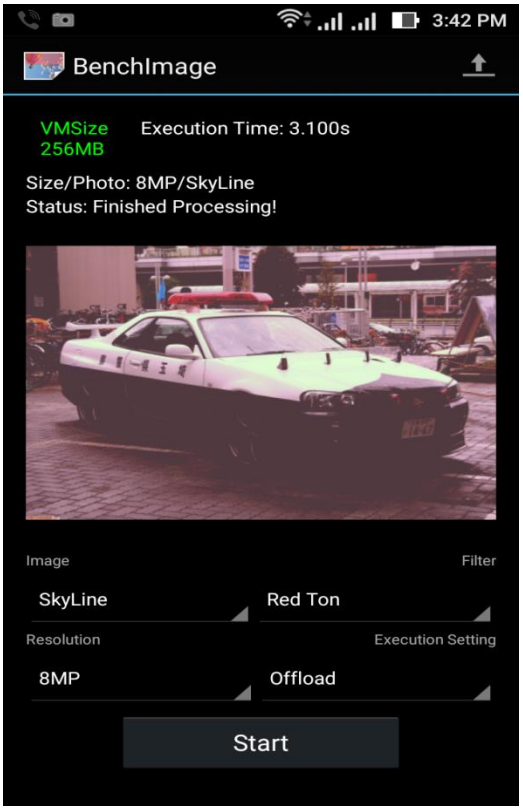

Figure 5.9 Screen shot for applying Red Ton effect in offloading

Table 5.3 Average Execution Time of Applying Red Ton Effect (seconds)

Execution Setting	**Image Resolution (MP)**				
	0.3	1	2	4	8
Local	0.777	2.683	4.871	9.48	20.23
Offloading	0.562	0.449	0.830	1.218	3.01
Performance Improvement	1.38X	5.97X	5.86X	7.78X	6.72X

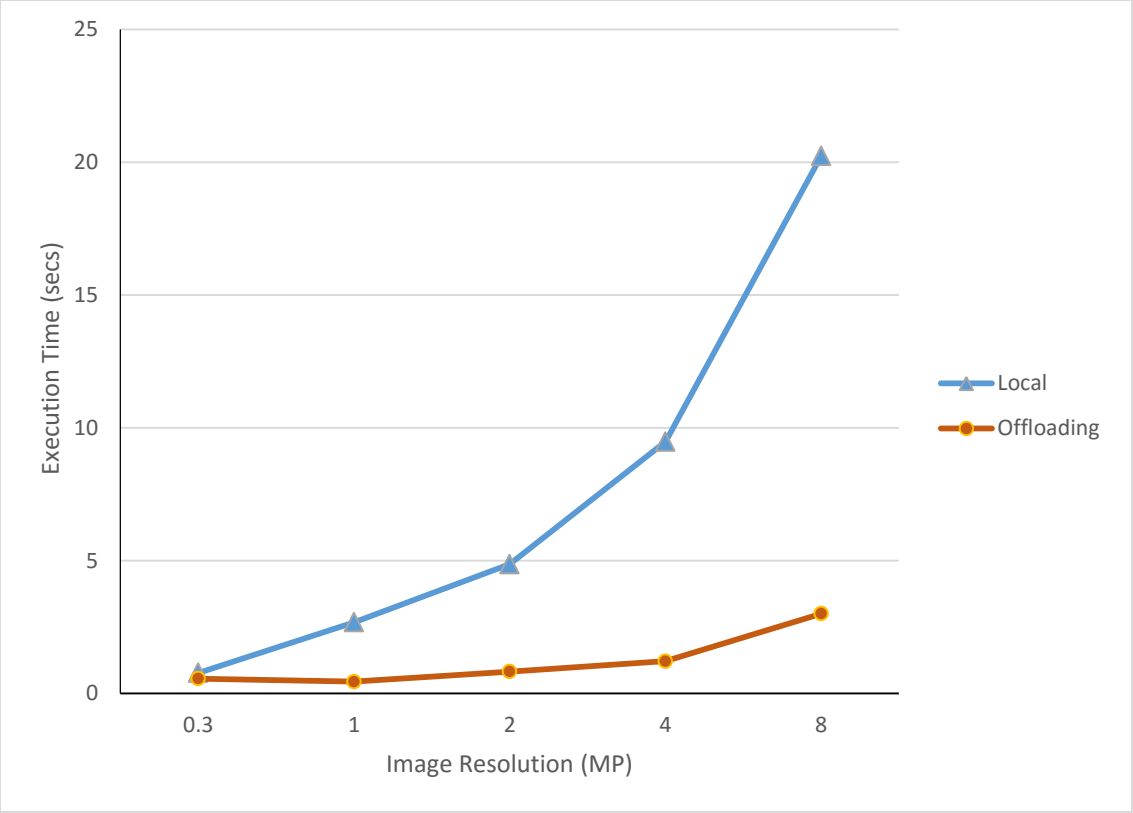

Figure 5.10 The result of applying a Red Ton effect

According to the above graphs that shows the results of evaluation, the execution time (or response time) of the mobile application in the local mobile device and remote cloudlet is almost the same for an image with lower resolution. The variation of the execution time rapidly increases as the resolution of the image increases. An image with lower resolution requires lower computation compared to an image with higher resolution due to the more number of pixels that needs processing. This can also be described as; the variation of the execution time increases as the computation requirement of the applications (or tasks) increases.

Moreover, according to the analysis of the code and the execution time, cartooning an image is more computation intensive than sharpening an image and sharpening is also more computation intensive than applying red ton into an image. The result shows that the performance improvement is better for the computation-intensive effects(filters). Those shows that the proposed computation offloading system is more beneficial for computation intensive mobile applications (tasks).

The size of the image that was used for the evaluation of the proposed computation offloading

system is 255 KB, 375 KB, 708 KB, 1.15 MB and 4.30 MB, for 0.3MP, 1MP, 2MP, 4MP and 8MP image resolution respectively. This indicates that the size of the image increases as the resolution of the image increases. According to the above graphs the proposed computation offloading system improves the performance of higher size images than the images with lower size. This is an indication for the proposed computation offloading system is more effective for mobile applications that requires higher input data size during execution.

Computation offloading is effective for mobile application which are computation-intensive and that requires low data transfer during execution[2]. However, the proposed computation offloading system is more effective for mobile applications that are computation-intensive and that requires higher data transfer. This is due to the partitioning algorithm that considers the execution time and the call relationship between each method in the mobile application. The proposed computation offloading system determines the offloading and integration point in the whole application, instead of examining each individual method. This reduced the data transfer between the mobile device and the remote cloudlet during the execution of the mobile application.

As it is detailly described in chapter 3 COSMOS[10] is a computation offloading framework that enhance the performance of mobile applications and reduces monetary cost of cloud services. The COSMOS framework achieved a speedup of 2.76X in stable and fast wireless network, which is much lower than what our computation offloading system have achieved. Even though a WLAN with 1.08Mbps to 3.20Mbps speed range cannot be categorized as a fast network[44], the proposed computation offloading system improved the performance of mobile applications with an average of 5.81X. This indicates that our proposed computation offloading system can optimize the performance of computation-intensive mobile applications almost twice as the COSMOS framework.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we have proposed and implemented a lightweight computation offloading system for mobile cloud computing that offloads applications at method level granularity. The primary aim of the proposed system is to optimize the performance of computation-intensive mobile applications. It also reduces the data transfer between the mobile device and the remote cloudlet during the execution of mobile applications.

Our proposed system considers the intercommunication between each method and their execution time for partitioning. Instead of examining each individual method for offloading, it determines the offloading and integrating points in the whole application using the critical path method which minimizes the execution time of the mobile applications. Besides this, the sophisticated components of the proposed computation offloading system (i.e. the application profiling and partitioning) that requires heavy computation are implemented at the server side to minimize overhead in the mobile side. A lighter and simple offloading engine is implemented at the mobile side. The offloading engine considers the quality of the wireless network during offloading decision.

We used the client/server communication model to implement the proposed computation offloading system and the Asynchronous Remote Procedure Calls (RPC) to enable communication between the mobile devices and the remote servers. Furthermore, the cloudlet computation offloading architecture was used to deploy the proposed computation offloading system. We implemented it on laptop computer configured as a cloudlet and on android mobile device. Finally, the proposed computation offloading system was evaluated and the obtained result shows that it can speedup mobile applications with an average of 5.81X. They also indicate that the proposed system is more effective for mobile applications that require heavy computation.

6.2 Future Work

The proposed computation offloading system has improved the performance of computation-intensive mobile application and its overall results have been positive. However, there are still some aspects of the system that require further research. In the future, we plan to extend our work to address the following:

- An intelligent offloading algorithm that considers the bandwidth and throughput of the network as a parameter for offloading decision.
- Automated integration of the proposed system with mobile applications to reduce the burdens on the developers.
- The proposed application partitioning algorithm can be more effective for parallel offloading, which is the simultaneous execution of the partitions in different server.
- Considering the security of the network and the cloud environment during offloading decision.
- Currently, the proposed computation offloading system supports only Android mobile applications and enabling the system to support multiple mobile platforms can widened the delivery of computation offloading services.

REFERENCES

[1] X. Liu, C.-W. Yuan, Y. Li, Z. Yang, and B. Cao, "A Lightweight Algorithm for Collaborative Task Execution in Mobile Cloud Computing," *Wirel. Pers. Commun.*, vol. 86, no. 2, pp. 579–599, 2016.

[2] H. Wu, "Analysis of Offloading Decision Making in Mobile Cloud Computing," 2015.

[3] H. FLORES, *Service-oriented and Evidence-aware Mobile Cloud Computing*. University of Tartu Press, 2015.

[4] R. Arora and A. Parashar, "Secure User Data in Cloud Computing Using Encryption Algorithms," vol. 3, no. 4, pp. 1922–1926, 2013.

[5] X. Chen, "Decentralized Computation Offloading Game For Mobile Cloud Computing," 2015.

[6] D. Kovachev, T. Yu, and R. Klamma, "Adaptive Computation Offloading from Mobile Devices into the Cloud," 2013.

[7] B. Chun and A. Patti, "CloneCloud : Elastic Execution between Mobile Device and Cloud," pp. 301–314, 2011.

[8] "A Framework for Energy-efficient Mobile Cloud Offloading," 2015.

[9] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," *MobiSys'10*, vol. 17, pp. 49–62, 2010.

[10] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "COSMOS : Computation Offloading as a Service for Mobile Devices," *Proc. MobiHoc*, pp. 287–296, 2014.

[11] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload : an efficient code partition algorithm for mobile cloud computing," pp. 80–86, 2012.

[12] E. Hyytia, "Offload (Only) the Right Jobs : Robust Offloading U sing the Markov

Decision Processes," 2015.

[13] Y. Lin, E. T. Chu, Y. Lai, and T. Huang, "Time-and-Energy-Aware Computation Offloading in Handheld Devices to Coprocessors and Clouds," pp. 1–13, 2013.

[14] H. Zhu, C. Huang, and J. Yan, "Vulnerability Evaluation for Securely Offloading Mobile Apps in the Cloud," pp. 108–116, 2013.

[15] N. S. Hauser, "COARA : Code Offloading on Android with RMI and AspectJ," *Tech. Inst. Technol.*, 2014.

[16] V. Vaishnavi and B. Kuechler, "Design Science Research in Information Systems," 2015. [Online]. Available: http://www.desrist.org/design-research-in-information-systems/. [Accessed: 12-Feb-2016].

[17] W. Zhang, S. Tan, F. Xia, X. Chen, and Z. Li, "A survey on decision making for task migration in mobile cloud environments," *Pers. Ubiquitous Comput.*, 2016.

[18] H. QIAN, "EXTENDING THE BATTERY LIFE OF MOBILE DEVICE BY COMPUTATION OFFLOADING," 2015.

[19] "Mobile Computing." [Online]. Available: www.tutorialspoint.com.

[20] C. De Alwis, "Mobile Cloud Computing." pp. 1–15.

[21] M. R. Momeni, "A Survey of Mobile Cloud Computing : Advantages , Challenges and Approaches," vol. 15, no. 4, pp. 14–28.

[22] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A Survey of Mobile Cloud Computing : Architecture , Applications , and Approaches," no. Cc, pp. 1–38.

[23] M. Wang, Y. Chen, and M. J. Khan, "Mobile Cloud Learning for Higher Education : A Case Study of Moodle in the Cloud."

[24] M. Furini, "MOBILE GAMES : WHAT TO EXPECT IN THE NEAR FUTURE," *GAMEON'2007*, 2007.

[25] K. Kumar and J. Liu, "A Survey of Computation Offloading for Mobile Systems," 2012.

[26] H. Wu, K. Wolter, and A. Grazioli, "Cloudlet-based Mobile Offloading Systems : a

Performance Analysis," pp. 2–3, 2013.

[27] T. Soyata, R. Muraleedharan-sreekumaridevi, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision : Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture."

[28] E. E. Marinelli, "Hyrax : Cloud Computing on Mobile Devices using MapReduce," vol. 0389, no. September, 2009.

[29] X. Wu, C. Xu, Z. Lu, Y. Jiang, C. Cao, X. Ma, and J. Lu, "CoseDroid : Effective Computation- and Sensing-offloading for Android Apps," 2015.

[30] R. Friedman and N. Hauser, "COARA : Code Offloading on Android with AspectJ," pp. 1–13, 2014.

[31] P. V. Krishna, "Learning Automata Based Decision Making Algorithm for Task Offloading in Mobile Cloud Department of Computing Science."

[32] M. Paul and S. Nir, "Scalable Resource Augmentation for Mobile Devices," 2014.

[33] M. Daf, "Scavenger : Transparent Development of Efficient Cyber Foraging Applications," pp. 217–226, 2009.

[34] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *J. Netw. Comput. Appl.*, vol. 48, pp. 99–117, 2015.

[35] S. A. Saab, F. Saab, A. Kayssi, A. Chehab, and I. H. Elhajj, "Sustainable Computing : Informatics and Systems Partial mobile application offloading to the cloud for energy-efficiency with security measures," *Sustain. Comput. Informatics Syst.*, vol. 8, pp. 38–46, 2015.

[36] M. Gordon, D. Jamshidi, and S. Mahlke, "COMET: code offload by migrating execution transparently," *Proc. 10th …*, pp. 93–106, 2012.

[37] M. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa : Enabling Interactive Perception Applications on Mobile Devices Categories and Subject Descriptors," *ACM*, 2011.

[38] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” *Proc. - IEEE INFOCOM*, pp. 945–953, 2012.

[39] H. Flores and S. N. Srirama, “Adaptive Code Offloading for Mobile Cloud Applications : Exploiting Fuzzy Sets and Evidence-based Learning,” *Proc. Fourth ACM Work. Mob. Cloud Comput. Serv. (MCS 2013)*, pp. 9–16, 2013.

[40] A. Dou, “Misco : A MapReduce Framework for Mobile Systems ∗,” 2010.

[41] K. Yang and S. Ou, “On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet,” no. January, pp. 56–63, 2008.

[42] R. Niu, W. Song, and Y. Liu, “An Energy-Efficient Multisite Offloading Algorithm for Mobile Devices,” *Int. J. Distrib. Sens. Networks*, vol. 2013, 2013.

[43] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-based Cloudlets in Mobile Computing,” vol. 1, 2006.

[44] S. Yang, X. Bei, Y. Zhang, and Y. Ji, “Application Offloading based on R-OSGi in Mobile Cloud Computing,” 2016.

[45] R. K. K. Ma and C. Wang, “Lightweight Application-level Task Migration for Mobile Cloud Computing,” *2012 IEEE 26th Int. Conf. Adv. Inf. Netw. Appl.*, pp. 550–557, 2012.

[46] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, “Mobile code offloading: From concept to practice and beyond,” *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, 2015.

[47] T. Nabi, E. Tilevich, and V. Tech, “Assessing the Benefits of Computational Offloading in Mobile-Cloud Applications,” pp. 17–24, 2015.

[48] K. W. D. Meng, Tianhui, Qiushi Wang, “Model-Based Quantitative Security Analysis of Mobile Offloading Systems Under Timing Attacks,” *Int. J. Simul.*, vol. 3, no. 3–4, pp. 1–3, 2015.

[49] V. Roubtsov, “Sizeof for Java Object sizing revisited,” 2003. [Online]. Available: http://www.javaworld.com/article/2077408/core-java/sizeof-for-java.html.

[50] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing : A survey,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[51] P. Nepali and O. L. Society, “Energy Efficient Method Level Offloading For Mobile Cloud Computing,” 2012.

[52] S. R. B, “A Report on RMI and RPC.”

[53] P. B. Costa, P. A. L. Rego, L. S. Rocha, and J. N. De Souza, “MpOS : A Multiplatform Offloading System,” pp. 577–584, 2015.

[54] “Rapid - Android Offloading Framework.” p. 4, 2016.

[55] “Java Call Graph Projects.” [Online]. Available: www.github.com. [Accessed: 08-Sep-2016].

[56] Q. Wang, “Restart in Mobile Offloading,” 2015.

[57] Joe, “Java Instrumentation,” 2013. [Online]. Available: http://javapapers.com/core-java/java-instrumentation/.

[58] S. Chiba, “Getting Started with Javassist.” [Online]. Available: http://jboss-javassist.github.io/javassist/tutorial/tutorial.html. [Accessed: 12-Jan-2016].

[59] M. A. Khan, “Scheduling for heterogeneous systems using constrained critical paths,” *PARALLEL Comput.*, vol. 38, no. 4–5, pp. 175–193, 2012.

[60] J. Zambrano, “Mobile Cloud Computing : Offloading Mobile Processing to the Cloud,” 2015.

[61] J. Schewe, “Image Sharpening-Make it Really Sharp.” [Online]. Available: http://www.schewephoto.com. [Accessed: 09-Nov-2016].

[62] Ember, “Color Tone An Image,” 2016. [Online]. Available: http://www.watermark-software.com/adjust-color-tone.html. [Accessed: 01-Feb-2017].

APPENDICES

Appendix A: Sample Code

```
// Java agent class of the application profiler

package yb.thesis.app.profiler;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.Instrumentation;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
import javassist.CannotCompileException;
import javassist.ClassPool;
import javassist.CtBehavior;
import javassist.CtClass;
import javassist.NotFoundException;

public class App_Profiler implements ClassFileTransformer {

   static List<Pattern> Include = new ArrayList<Pattern>();
   static List<Pattern> Exclude = new ArrayList<Pattern>();

   public byte[] transform(ClassLoader cl_Loader, String cl_Name, Class cl,
         java.security.ProtectionDomain domain, byte[] bytes) {
      boolean instrumentClass = false;
```

```
        String class_name = cl_Name.replace("/", ".");
        for (Pattern ptr : Include) {
            Matcher m = ptr.matcher(class_name);
            if (m.matches()) {
                instrumentClass = true;
                break;
            }
        }
        for (Pattern p : Exclude) {
            Matcher m = p.matcher(class_name);
            if (m.matches()) {
                err("Not instrumenting class: " + class_name);
                instrumentClass = false;
                break;
            }
        }
        if (instrumentClass) {
            return instumentClass(cl_Name, bytes);
        } else {
            return bytes;
        }
    }

public static void premain(String argument, Instrumentation instrumentation) {
        if (argument == null) {
            err("Missing configuration argument");
            return;
        }
        err("Argument is: " + argument);
        String[] tokens = argument.split(";");
        if (tokens.length < 1) {
            err("Missing delimeter ;");
```

```
            return;
        }
        for (String token : tokens) {
            String[] args = token.split("=");
            if (args.length < 2) {
                err("Missing argument delimeter =:" + token);
                return;
            }
            String argtype = args[0];
            if (!argtype.equals("incl") && !argtype.equals("excl")) {
                err("Wrong argument: " + argtype);
                return;
            }
            String[] patterns = args[1].split(",");
            for (String pattern : patterns) {
                Pattern p = null;
                err("Compiling " + argtype + " pattern:" + pattern + "$");
                try {
                    p = Pattern.compile(pattern + "$");
                } catch (PatternSyntaxException pse) {
                    err("pattern: " + pattern + " not valid, ignoring");
                }
                if (argtype.equals("incl"))
                    Include.add(p);
                else
                    Exclude.add(p);
            }
        }
        instrumentation.addTransformer(new App_Profiler());
    }
```

```
private byte[] instumentClass(String name, byte[] b) {

   ClassPool pool = ClassPool.getDefault();
   CtClass clazz = null;
   try {
      clazz = pool.makeClass(new ByteArrayInputStream(b));
      if (!clazz.isInterface()) {
         err("Instrumenting class: " + name);
         CtBehavior[] methods = clazz.getDeclaredBehaviors();
         for (int i = 0; i < methods.length; i++) {
            if (!methods[i].isEmpty()) {
               instrumentMethod(methods[i], clazz.getName());
            }
         }
         b = clazz.toBytecode();
      }
   } catch (CannotCompileException e) {
      e.printStackTrace();
      err("Cannot compile: " + e.getMessage());
   } catch (NotFoundException e) {
      e.printStackTrace();
      err("Cannot find: " + e.getMessage());
   } catch (IOException e) {
      err("Error writing: " + e.getMessage());
   } finally {
      if (clazz != null) {
         clazz.detach();
      }

   }

   return b;
```

```
    }

    private void instrumentMethod(CtBehavior method, String className)
            throws NotFoundException, CannotCompileException {
        String name = className.substring(className.lastIndexOf('.') + 1, className.length());
        String methodName = method.getName();
        if (method.getName().equals(name))
            methodName = "<init>";
        method.insertBefore("yb.thesis.app.profiler.MethodTimer.push(\"" + className
                + ":" + methodName + "\");");
        method.insertAfter("yb.thesis.app.profiler.MethodTimer.pop();");
        }
    private static void err(String msg) {
        System.err.println("[App-pro] " + msg);
    }
}
```