Exact and Tradeoff Decision Making for Code Offloading in Mobile Cloud Computing

by

Mahbub E Khoda

Class Roll: SH–246

Exam Roll: Curzon–1906

Registration No: H–1273

M.Sc. Session: 2011–2012

A Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF DHAKA

June 2014

Declaration of Authorship

I, Mahbub E Khoda, declare that this thesis titled, 'Exact and Trade-off Decision Making for Code Offloading in Mobile Cloud Computing' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Countersigned: _____

(Dr. Md. Abdur Razzaque)
Supervisor

Signed: _____

(Mahbub E Khoda)
Candidate

UNIVERSITY OF DHAKA

Abstract

Faculty of Engineering and Technology
Department of Computer Science and Engineering
University of Dhaka

Master of Science

by Mahbub E Khoda

Due to the significant advancement of Smartphone technology, the applications targeted for these devices are getting more and more complex and demanding of high power and resources. Mobile cloud computing allows the Smartphones to perform these highly demanding tasks with the help of powerful cloud servers. In this paper, we developed an intelligent code offloading system ExTrade that takes *exact* and *tradeoff* decisions for code offloading from a mobile device to cloud server. We used Lagrange multiplier method to define two types of decision making regions for code offloading. We developed a metric for tradeoff decision making that can maximize energy saving while maintaining QoS requirements of user applications. We used statistical regression technique to estimate the execution time of a task eligible for offloading. We evaluated the performances of the proposed ExTrade system in a testbed implementation and the results show that it outperforms the state-of-the-art methods in terms of accuracy, computation and energy saving.

Acknowledgements

Thanking is the best way to appreciate own-self for performing a great talk. The reason I can claim that I am successful in completing my work with such ease is that almighty ALLAH gave me the ability, capability, chance, cooperating hands, and enough stamina to walk on this path. ALLAH enriched me with very good luck on this path of work. The reason I am talking about luck is that I had my respected teacher as my supervisor in this work and I would like to take the opportunity to express my sincere gratitude and indebtedness to my respected supervisor Dr. Md. Abdur Razzaque. Although he is an internationally recognized person with always loaded with administrative works, he gave me more than enough time in this work. He not only gave me time but also proper guidance whenever I faced with some difficulties. He kept his door always open for me in all the days, all the time even in the holidays. He gave me the access to contact with him directly in his cell phone regarding the work. I still remember those memorable days, when I returned to my home at around 9 p.m. from the department after fruitful discussion with him and he is the person who gave me such enormous support. I must also be thankful to my teachers who helped me in a number of ways by providing various resources and moral support. Finally, I would like to thank my mother, my brothers and sisters, and my friends for their unconditional love and selfless support.

Contents

List of Figures

List of Tables

Abbreviations

MCC	**M**obile **C**loud **C**computing
SLA	**S**ervice **L**evel **A**ggrement
VM	**V**irtual **M**achine
KKT	**K**arush **K**uhn **T**ucker
EH	**E**xecution **H**andler
HCLD	**H**eavy **C**omputation **L**ess **D**ata
HCHD	**H**eavy **C**omputation **H**eavy **D**ata

Symbols

S_f	Speed factor
λ	Langrange multiplier
α	Tradeoff weight
t_m	Execution time in the mobile device
t_c	Execution time in the cloud server
p_m	Average power consumption in the computation mode
p_i	Average power consumption in the idle mode
p_t	Average power consumption in the transmission mode
G_{th}	Tradeoff threshold

Chapter 1

Introduction

1.1 Overview

We are now living in an era of smart devices. According to a research conducted by Pew Research Center in 2013, 56% of American adults own a smartphone of some kind [1]. Another research commissioned by New Relic, a web application performance company, revealed that 1.3 million Android devices are activated every day [2]. These devices are equipped with various functionalities such as GPS, WiFi, cameras, sensors, good level of storage and processing speeds that lead them installed with wide range of applications. More interestingly, the applications targeted for these devices are getting more and more complex with the number of users and the increasing capabilities of the devices. And in recent years, these applications have started becoming abundant in various categories such as image processing, gaming, participatory sensing, real-time monitoring, social networking, travel and news, etc [3]. These sophisticated and resource-hungry applications are increasingly posing requirements especially for more energy and computation power. Since, mobile devices are inherently constrained by processing power and energy [4, 5], the topic of saving mobile power and minimizing CPU consumption of mobile devices has got attraction of researchers in this field.

Mobile cloud computing (MCC) [6, 7, 8] enables a resource constrained mobile device to perform complex application with the power of cloud computing. The task that needs to perform complex computation and is not feasible to be executed

on the mobile device could be migrated to the cloud server and the execution of the task is performed on the powerful machine of the cloud rather than on the mobile devices. This process of task migration is known as code/task offloading in the literature. The challenge of this technique is determining when a task should be migrated to the cloud in order to be get benefit from the immense power of the cloud computing. Another import aspect is to estimate the execution time of the task that is under consideration for offloading.

In this thesis, we address the problem of code offloading decision making and task execution time estimation. We design a system namely, ExTrade to analyze the condition under which a task should be offloaded to the cloud server and make decisions accordingly. Our proposed system also uses statistical regression based for estimating the execution time of task. We show that this ExTrade system performs better under dynamic condition of the task execution environment.

1.2 Motivation

While the growth in smartphone technology enables the application developers to build sophisticated applications for the users, the limited resource of the mobile devices becomes a bottleneck for these applications [9]. Especially, the limited battery power of the mobile devices becomes a headache for the application users as the sophisticated application targeted for these devices need complex computations and demands high amount of energy from the mobile device in turn [10].

On the other hand, the advancement of cloud computing [11, 12] has dramatically increased the capability of resource constrained devices to perform at an unprecedented level [13]. The resource constrained mobile devices can perform complex tasks with the help of cloud computing with improved computational performance and energy efficiency. To get the benefit of the immense power of cloud computing for performing computation and energy intensive task in the mobile device it is important to know when a task should be delegated to the cloud server in order to improve application response time and energy consumption of the mobile device . This thesis work focuses on finding the necessary conditions that a task should

meet before getting executed on the cloud server instead of getting executed on the local device.

1.3 Mobile Cloud Computing

Mobile cloud computing at its simplest, refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from the mobile devices and into powerful and centralized computing platforms located in clouds, which are then accessed over the wireless connection based on a thin native client.In other words, mobile cloud computing is the combination of cloud computing and mobile networks to bring benefits for mobile users, network operators, as well as cloud computing providers [14, 15]. The ultimate goal of MCC is to enable execution of rich mobile applications on a plethora of mobile devices, with a rich user experience [16, 17].

1.3.1 Why Mobile Cloud Computing?

Mobile devices are inherently constrained by limited battery life and faces many resource challenges (limited storage, inconsistent network bandwidth etc.). On the other hand, cloud computing offers advantages to users by allowing them to use infrastructure, platforms and software by cloud providers at low cost and elastically in an on-demand fashion. Mobile cloud computing provides mobile users with data storage and processing services in clouds, obviating the need to have a powerful device configuration (e.g. CPU speed, memory capacity etc), as all resource-intensive computing can be performed in the cloud [18].

1.3.2 Mobile Cloud Computing Architecture

Figure 1.1 shows the basic architecture of mobile cloud computing environment. The basic operations of the MCC architecture is described below:

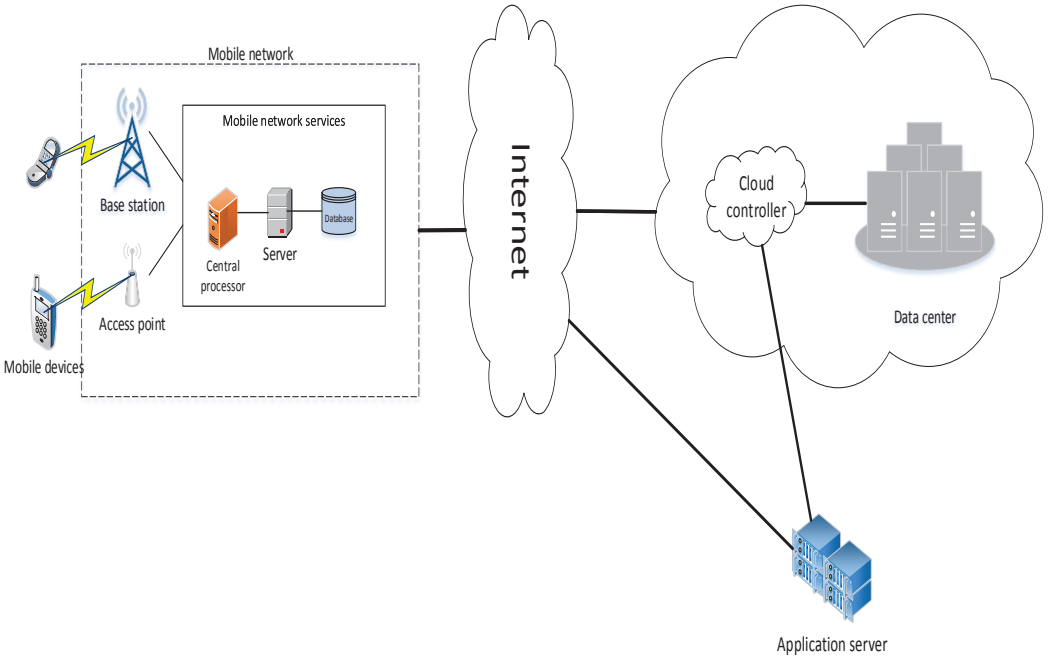


FIGURE 1.1: Mobile cloud computing architecture

- Mobile devices are connected to the mobile networks via base stations that establish and control the connections and functional interfaces between the networks and mobile devices.
- Mobile users' requests and information are transmitted to the central processors that are connected to servers providing mobile network services.
- The subscribers' requests are delivered to a cloud through the Internet.
- In the cloud, cloud controllers process the requests to provide mobile users with the corresponding cloud services.

1.3.3 Advantages of MCC

- **Extending battery lifetime:** Computation offloading migrates large computations and complex processing from resource-limited devices (i.e., mobile devices) to resourceful machines (i.e., servers in clouds). Executing these heavy computational task on the remote servers can save energy significantly. Many mobile applications take advantages from task migration and remote processing.

- **Improving data storage capacity and processing power:** MCC enables mobile users to store/access large data on the cloud. Since the data now are stored on the cloud, mobile applications are no longer constrained by storage capacity.
- **Improving reliability and availability:** Keeping data and application in the clouds reduces the chance of data loss from the mobile devices.
- **Scalability:** Since cloud computing provides the users with on demand resource allocation the mobile applications can be performed and scaled to meet the unpredictable user demands.

1.3.4 Basics of Computation Offloading from Mobile Device to Cloud Server

Mobile cloud computing enables resource constrained mobile devices to execute more demanding application by executing computationally intensive parts of an application on a remote cloud server. This technique of executing the more resource demanding part of an application on the cloud server instead of the local device is known as code offloading. There are two major types of offloading process found in the literature namely *VM migration* and *code partitioning*.

1.3.4.1 VM Migration

In this technique the state of the total virtual machine from the mobile device is transferred to the cloud server. This process basically recreates the mobile execution environment on the cloud server with more computation power. Then the application is executed to that powerful virtual machine. Executing mobile application on such powerful virtual machines reduces the time of execution and the output of the execution is produced more quickly. But major problem of this technique is that it needs a huge amount of data transfer for the offloading process. Because transferring the whole virtual machine state to the cloud server requires a huge number of information to be exchanged between the cloud server and mobile

device. For this reason this technique of offloading code to the cloud server is no considered to be a feasible approach in mobile cloud computing.

1.3.4.2 Code Partitioning

In this technique the application code running on the mobile device is partitioned into two categories. One category of the partitioned code gets offloaded to the cloud server and other category is executed in the local device. The offloaded portion of the code basically requires more computation time and demands more energy from the mobile device. The advantage of this technique is that it gets rid of the problem of huge data transfer as it was needed by the VM migration technique. Moreover, in this technique a copy of the device or application is generally kept on the cloud server. So offloading in this technique often only requires exchanging the application and task ID and the data that are essential for executing the task and are not already present in the cloud server.

1.4 Challenges in Mobile Cloud Computing

Though using the improved computation power of cloud computing the mobile devices can perform complex computations by migrating task to the cloud servers, the process of migrating a task is not that straight forward. Studies about mobile cloud computing have primarily focused on the following issues:

1.4.1 Offloading Code to The Cloud Server

To get the best from the offloading technique it is important to decide when to offload a task and when not to. To always offload a task to the cloud server would be a naive approach because if a task is not computationally intensive and requires a large amount of data transfer for offloading, this may lead to higher energy consumption and response time[19]. To make the offloading decision correctly, we need to specify the conditions a task should satisfy to before being offloaded. Informally, this condition could be stated as, if offloading a task improves the

application response time and saves energy consumption from the mobile device the task should be offloaded. But, in order to check if the condition is satisfied a proper technique is needed that would solve the condition checking problem.

1.4.2 Estimating Task Execution Time

1.4.2.1 Motivation

Before making the offloading decision it is essential to know the estimated execution time of the task in the mobile device. Based on this estimation the cost for offloading a task to the cloud server is calculated and comparing this cost with the local execution cost the code offloading decision is made. But the process of estimating the task execution time is not that straight forward due to various factors affecting the execution time of the task. Such as, user inputs to the task, CPU speed of the machine executing the task, the current load of the cpu etc. Therefor, the process of estimating the execution time of the task needs to be studied carefully.

1.4.2.2 Task Execution Time Estimation Methods

A proper technique for estimating the task execution time, which plays an important role in the decision making, is absent in the previous works. Some previous works assumes the execution time information is already known to the system. This assumption is not practical and not always true. Other works uses global averaging on some previous observations to estimate the execution time of the task. This technique can predict the execution time of a task effectively if the characteristics of the task remains the same over the time. But, for the task that behaves dynamically depending upon various parameters such as, user inputs; the averaging technique cannot be used as effectively. A new context aware approach was introduced in [19] where the time and location of the device was taken into consideration when estimating the task execution cost. The work argued that for a given location and a given time the network condition remains the same. While this argument might be true to some extent, still it cannot predict the

execution time of the task when the application usage behavior does not remain consistent. In this work, we use technique of statistical regression[20] to estimate the task execution time that estimates the execution cost of the task with respect to some modeling parameters such as user inputs, current CPU load, etc. And it uses a residue factor that compensates for the error incurred in the estimation process due to the unmodeled parameters. Our experimental results show that this technique of task execution time estimation performs better than the existing techniques under dynamic conditions.

1.4.2.3 Offloading Granularity

The current trend of Smartphone application development is to keep the user interface responsive to the application and it is considered as a quality metric for mobile application [21]. In fact, the Android programming guideline [22] encourages the application developers to execute computation intensive tasks in a separate worker thread and keep the main thread or user interface responsive to the user. Therefore, instead of method offloading, we go for thread level code offloading in this work, which is of more practical. In the following, we use the terms *code*, *task*, and *thread* interchangeably.

1.5 Proposed Solution Model

The challenges in MCC have been given importance in our project and a solution model has been designed in our proposed system. The offloading decision making and estimating the task execution time problem can be solved by our ExTrade system in an efficient way. We prove that our solution model yields better performance for the problems of MCCs. In this solution model the decision making process is done using a non-linear optimization method and the task execution time is predicted using a statistical regression technique and this solution methodology performs better in the dynamic execution environment of the task.

The Figure 1.2 shows the overview of the solution model of our ExTrade system. Here, the decision maker or the execution handler(EH) plays the most important

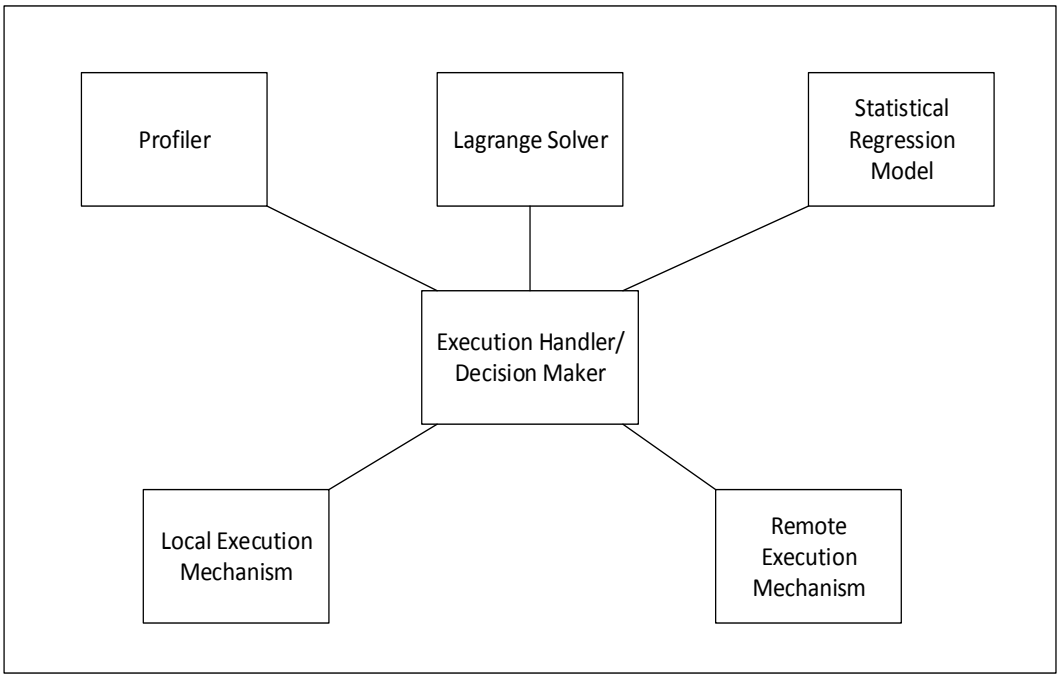


FIGURE 1.2: Solution model of ExTrade system

role in our solution model which is responsible for making the decision about where the task should be executed (in cloud or in mobile). The solution model is described below briefly:

1.5.1 Execution Handler

When a task is about to execute the execution handler checks to see if offloading the task would benefit the execution of the application by using the information provided by the profiler, Lagrange solver and the statistical regression model. If offloading the task is beneficiary to the application, remote execution request of the task is issued to the cloud server, otherwise the local execution is performed.

1.5.2 Profiler

The current condition of the mobile network and the cloud server load plays an important part regarding the performance of the application. The profiler keeps

updated information about the current available network bandwidth and the current load of the cloud server provides these information to the execution handler when needed.

1.5.3 Lagrange solver

It is important to know whether offloading a task to the cloud server will save computation or energy before making the offloading decision. The Lagrange solver finds the maxima and minima value of the offloading cost function and defines the regions where a task should be offloaded.

1.5.4 Statistical regression model

The decision of offloading a task to the cloud server is made based on the estimated task execution time on the mobile device. Statistical regression model in our ExTrade system predicts the execution time of a task considering the dynamic behaviour of the application user and task execution environment.

1.6 Contribution

The contribution of our work is that we define the decision regions(i.e. exact and tradeoff) using the Lagrange Multiplier method for making the code offloading decision. We designed a tradeoff metric for the region where exact decision cannot be made and proposed a decision mechanism in that region. We propose a statistical prediction technique to estimate the task execution time on the mobile device that takes into consideration the dynamic behaviour of the application user and the dynamic change in environment. The following summarizes the contribution of this thesis work:

- We develop an exact and tradeoff decision making system for code offloading in MCC, ExTrade, that determines code offloading decision regions: *exact*

and *tradeoff* using Lagrange Optimization, a lightweight non-linear optimization method.

- We design a code offloading decision metric that effectively helps us to tradeoff between the computation time and energy costs.
- We propose a statistical regression based solution for estimating the execution time of the task that is highly accurate than time average functions.
- We implement our proposed ExTrade in a test-bed and the experimental results show that our mechanism achieves approximately 62% more prediction accuracy, in a dynamic application environment, than a number of state-of-the-art techniques

1.7 Organization of The Thesis

Chapter 2, titled 'Background and Motivation' discusses the existing work on the issue of code offloading and task execution time estimation. In chapter 3, we describe the system and code architecture of the ExTrade system and the functionality of its components. In chapter 4, we describe the code offloading decision making technique and the task execution time prediction mechanism. Chapter 5 shows the implementation result of ExTrade system and compares the obtained result with a number of state-of-the-art works and chapter 6 concludes the paper.

Chapter 2

Background and Motivation

In the previous chapter, we have introduced the idea of mobile cloud computing and the process of code offloading process. Offloading the computation intensive part of an application the cloud server enables the resource constrained devices to execute more complex application in an energy and computation efficient manner. The importance of estimating the execution time of a task on making the offloading decision is also discussed there.

In this chapter, we give an overview on the motivation of code offloading process in mobile cloud computing. We also discuss some offloading techniques and architecture that have been introduced in earlier works.

2.1 Introduction

The advancement in the smartphone technology enabled the application developer to build more complex applications for the smartphone users. But mobile devices are constrained by many resource challenges (limited battery power, limited memory, network etc.). As a result the power of cloud computing is exploited to let these resource constrained mobile devices to run complex application in order to obtain improvement on the application performance and energy consumption. Significant number of methodologies and frameworks are designed to enable the resource constrained mobile devices to use the immense power of cloud computing.

Recently several task migration techniques for mobile cloud computing have been designed but most of them do not consider the computation and energy aspect together when making the offloading decision. On the other hand the estimation of the execution time of the task that is under consideration for offloading plays an important role in making the offloading decision. But very few of the work present in the current literature gives adequate importance on this issue. In this chapter, the subsequent sections are organized as follows. In section 2.2 we discuss a number of code offloading technique in MCCs. After that in section 2.3 we discuss the technique of task execution time estimation.

2.2 Code Offloading

The technique of offloading code from a resource constrained device to a resource reach architecture, often known as surrogates, have been studied in various works in the literature. Several frameworks [23, 24, 25, 26] have been proposed and a number of techniques have been introduced for the offloading process[4, 19, 21, 27, 28, 29, 30, 31, 32]. Mainly two approaches of the offloading process have been found in the literature. Each of them is discussed below:

1. VM migration: In this method of offloading the whole state of the virtual machine of the mobile device is transferred to the cloud server[33]. The server then recreates the copy of the virtual machine on its machine and execute the task. A modified version of it is was introduced by Satyanarayanan[30] which transferred a smaller state of VM to the server.

2. Code partitioning: In this approach the execution of intensive part of the application is done on the cloud server. Generally in this method a device clone or application clone is already present at the cloud server. The cloud server synchronizes with the mobile device through some mechanism to keep the updated information of the applications that are residing on the mobile device.

Our work takes on the second approach since offloading VM to the server requires a huge amount of data and state transfer and therefore is not a practical solution.

Offloading a portion of the code, that requires more computation power and energy, can overcome the problem of huge data transfer and achieve fair amount of savings in energy and resource in mobile devices. In literature several methodologies have been introduced that discuss about the process of offloading code from the mobile device to the cloud server.

2.2.1 Cloudlet

The author of Cloudlet[4] are among the very first to investigate the task migration process from a mobile device to a cloud server. They introduced a technique where a nearby resource rich computer, which might be disposed on a nearby coffee shop, acts as cloud provider and a smartphone connects to it over a wireless LAN. The rationale for this technique as described in the paper is that connecting with an actual cloud provider from the smartphone suffers from a higher latency and lower bandwidth because an actual provider is likely to be situated far away from the location of the smartphone. Connecting through a cloudlet located near by mitigates those problems.

2.2.2 MAUI

A number of optimization technique have been studied in the literature for offloading code in MCCs. The author of MAUI[27] first introduced a global optimization technique on the method call graph to offload methods to the cloud server for optimizing the energy consumption of a mobile device. They argued that considering each of the method individually cannot optimally solve the offloading problem since it cannot capture the whole picture of method execution. They showed that offloading some method may in effect execute more methods on the cloud server if that method invokes other methods. So offloading that method will save the cumulative cost of those methods. MAUI makes the offloading decision by analyzing the trade off between the energy consumed by the remote execution and local execution. Although in many cases offloading in this techniques often leads to performance improvement but ignoring the computational aspect of the task could lead to situations where the performance of the application might suffer.

2.2.3 CloneCloud

The analysis and partitioning of the application binary is addressed in this paper[28]. The authors of CloneCloud introduced a technique where the application binary is first marked to define various portions of the code that could be offloaded to the cloud server. Then some random set of inputs is generated prior to the execution. Then based on each randomly generated input a linear optimization solver was run to partition the application binary into two sets. One of the sets is to be executed on the cloud server and the other is to be executed on the local device. The partition information for these randomly generated inputs are saved in a database and the application is run the input to the application is matched with this database that the partitioning information obtained from the database is used for code offloading. This technique depends heavily on the randomly generated input sets for making the offloading decision. The randomly generated input sets prior to the execution of the application cannot cover all the dynamic conditions that could arise on the course of the execution of the application. Generally the mobile application user behaves dynamically and if the user gives input to the application that cannot be matched to any of the condition saved in the database this technique of offloading would drastically fail.

2.2.4 ThinkAir

The problem of previous works in MAUI and CloneCloud was first attacked by the authors of [29]. They improved the technique of code offloading by adopting an online method level offloading. The code architecture of ThinkAir requires the application developer to annotate the methods that could be offloaded to the cloud server. These methods must satisfy some constraints as mentioned in the paper. They also designed their custom compiler to convert a task for remote execution. This paper makes the offloading decision based on a set of previous execution history. The previous remote and local execution information is saved in the database. When making the offloading decision the average of the previous local and remote execution time is taken estimate the cost of remote and local execution. Comparing these execution cost this technique makes the

code offloading decision. The problem with this approach is that by averaging the previous execution history cannot make accurate prediction on the task execution cost if the behavior of the task is not consistent. Since the offloading decision mainly depends on this estimation of the execution cost inaccurate estimation can lead to poor choices.

2.2.5 CADA

The context-aware approach for offloading code the cloud server from the mobile device was introduce in the paper [19]. The author of CADA[19] argued that the mobile device user moves within a certain set of location. For example a student is expected to be in his school campus on class times or an employee is expected to be at his office on office time. They also argued that the move of a mobile user is similar on a given time of the day. For example an employee will use the same road on his way from home to office. They assumed that the network condition remains the same at given time and location. The saved the past history of a task execution on a given time and location in a database. Whenever a task is considered for offloading the local and remote execution cost of the task is estimated using the past history of that task on that particular time and location. Comparing this estimated cost the offloading decision is made. While in some of the cases this technique can estimate the execution cost of a task effectively, the work ignores dynamic behavior of the application user. If the application user does not behave in consistency with the time and location of the mobile device this technique cannot accurately estimate the task execution cost and fails to make correction decision on code offloading.

2.3 Estimating Task Execution Time

Another important aspect of decision making is the estimation of the task execution time [34]. Some of the previous works assumed that this information is already present in the system which is not practical. Other works keeps historical data and uses simple time average to predict the execution time. But regarding

different types of applications targeted for today's smartphones and the dynamic behaviour of the users of those applications this technique might lead to non-optimal decision. We use statistical regression technique[35] to estimate the task execution time. [20] introduced the model where the execution time of a task could be predicted using both non-parametric and parametric regression technique. The parametric part was used for scheduling a task among several machines. Our work deals with the problem using the non-parametric regression technique.

2.4 Tradeoff Analysis

There are situations where offloading a task to the cloud server might take more time than executing the task on the mobile device. But since the mobile device consumes most of its energy while being in the computation mode and offloading a task to the cloud leaves the mobile in idle mode, offloading such task would still save energy from the mobile device with a cost of degraded performance. But if the additional delay incurred in such scenarios is tolerable to the application user (i.e. does not violate the QoS requirement), offloading such tasks would still be beneficiary. Hence, a proper tradeoff metric needs to be designed so that accurate decisions can be made in such scenarios.

The analysis of tradeoff between energy and computation saving was introduced in[36]. Where three regions for making offloading decision is identified. For making decision in the tradeoff region a combined tradeoff metric was designed. But the combined metric designed for the tradeoff region is misleading for real valued variables due to the power function used in the metric. In this work we developed a decision metric that gets rid of the inconsistencies present in the earlier work and allows us to make proper decisions.

2.5 Discussion

The key characteristics of some of offloading technique in MCC is summarized in Table 2.1. Considering the dynamic user behavior is the most crucial factor when

Table 2.1: Characteristics of offloading techniques in MCC

Offloading technique	Decision making process	Execution cost estimation
MAUI[27]	Linear optimization	Assumed to be present
CloneCloud [28]	Linear optimization	Assumed to be present
ThinkAir [29]	Runtime calculation	Global averaging
CADA [19]	Runtime calculation	Local averaging
ExTrade	Runtime calculation	Statistical regression

estimating the task execution cost. But none of the offloading technique considers this when the task execution cost is calculated. Except the ExTrade approach, the other approaches do not address the dynamic factors affecting the execution cost of a task. ExTrade approach estimates the execution cost of the task using statistical regression technique that models the dynamic user behavior when the offloading is being made and makes more correct decisions than the other techniques.

2.6 Conclusion

Though many offloading techniques have been developed, none of them yet gave proper importance to the dynamic usage behavior of the application when estimating the task execution cost. Since, the estimation of task execution cost plays an important role in making the code offloading decision our proposed work considers the impact of dynamic user behavior and makes use of statistical regression technique that models the dynamic input of the application user when making the estimation of the task execution cost.

Chapter 3

Design Goals and Architecture

In this chapter, we discuss the directional design goals, system architecture and code architecture of our proposed ExTrade system. We also describe the work flow of our ExTrade system.

3.1 Introduction

Different approaches are present in the current MCC scenario for migrating task from the mobile device to the cloud server. One could migrate the whole VM state to the cloud or one could delegate some part of the application to the cloud server. Our proposed ExTrade system keeps a device clone on the cloud server that is capable of executing the application code present in mobile device. When a task is to be offloaded the remote server is notified with the task execution request by the application id and the task id (in our case it is the tread id).

In this chapter, we first discuss the assumption about the code organization and connectivity characteristic of the mobile device. After that we discuss about the code and system architecture of our proposed work. And then we describe the work flow of our proposed ExTrade system.

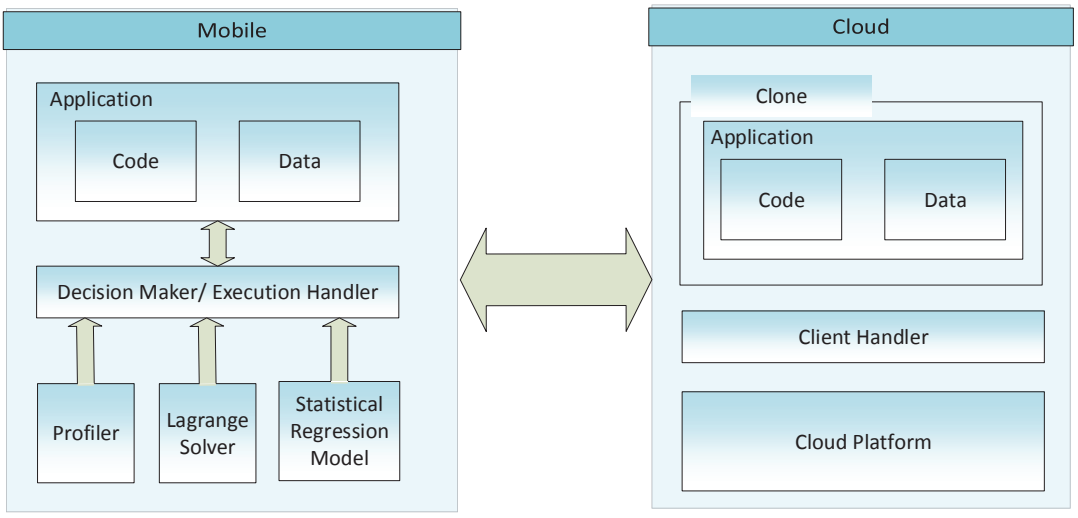


FIGURE 3.1: System architecture of ExTrade

3.2 Assumptions

Our technique makes the assumption that that the intensive part of mobile applications is executed in a worker thread [22]. For simplicity, we also assume that upon completion of a thread, a callback method is invoked. We also assume that the Smartphones are connected to the Internet by WiFi or 3G or 2G technology. The design goals, system architecture and code architecture for our proposed code offloading system, ExTrade, are described below.

3.3 Design Goals

To achieve improvement in computation performance and energy efficiency by offloading a task from the mobile device to cloud server our work proceeds keeping the following design goals into focus:

- **Performance improvement**: Our system aims to improve on both the aspects of computational performance and energy-efficiency of the mobile device by using the power of cloud servers. When improvement is possible only for one aspect, the system looks for an acceptable compromise in between the computation time and energy expenditures.

- **Adaptation to dynamic network changes**: Mobile network is used to send the remote execution request to the cloud server and rapid and dynamic change is one the main characteristics of the mobile network. Our proposed ExTrade aims to adapt quickly and effectively to the changes in mobile network environment.

- **Adaption to dynamic computation load of cloud server**: The cloud server response time for a task is affected by the current computation load of a cloud server. The less the load of the cloud server, the better is the response time. So, the offloading decision could be made more effectively by considering the dynamic load of the cloud server.

- **Enhancing QoE**: The Quality of Experience (QoE) of a mobile application user can be enhanced by utilizing the power of mobile cloud computing. The response time of an application could be improved dramatically by offloading a heavy computation task to a fast cloud server, which in turn improves the QoE. Furthermore, saving power by executing a task in the cloud server enables us to extend the battery life of a mobile device. Today's Smartphone users suffer mostly from poor battery lifetime. As a result, 22 million dollars are spent every year in electric utility cost due to keeping cell phones plugged into power sources to maintain full charge [10, 37]. By extending the battery lifetime of Smartphones a better QoE for the mobile application users can be achieved.

3.4 System Architecture

Figure 3.1 depicts the system architecture for our work. On the Smartphone, along with the application code and data, the different components of offloading decision making are present. On the cloud server, where the requested task is executed, along with the clone of the Smartphone, a client handler is present to handle request from the Smartphone. What follows next, we describe each of the major components of the Smartphone and the cloud server.

3.4.1 Execution Handler

Execution handler or the decision maker makes the decision about local or remote execution of a thread. When a thread is about to be executed, its eligibility for offloading is checked first. If the thread is offloadable, the control is handed over to the execution handler. The execution handler then obtains information from the profiler, statistical regression model and the lagrange solver. Considering all the available information the execution handler decides whether offloading the task would be profitable or not. Should the task be executed remotely, the remote execution request is made; otherwise, normal execution is resumed.

3.4.2 Profiler

Profiler keeps updated information about the dynamic cloud server and environmental conditions. It periodically communicates with the cloud server for connectivity information along with that it obtains other necessary updates from the cloud server. Following are the major information that the profiler keeps track of:

- **Transit delay**: Profiler provides with the information of data transmission delay between the mobile and the cloud server based on the available network bandwidth and data size. Since the available bandwidth of a mobile network is highly time-varying, the updated information is crucial when calculating the remote execution cost that drives the offloading decision.
- **Computation load of cloud server**: The response time of the cloud server for a task depends highly on the current load of the cloud server. When calculating the cost of remote execution, speed factor of the cloud server compared to the mobile device is considered to obtain the amount of improvement on the computation time and energy consumption. But, the load of the cloud server changes from time to time based on the job sizes on its queue, the number of requests from the clients, etc. Therefore, accurate measurement of speed factor is needed make the optimal offloading decision. In our proposed ExTrade, when periodically communicating with the cloud server the profiler obtains the current server load information.

3.4.3 Lagrange Solver

The Lagrange solver provides with the basis for making the offloading decision based on the estimated execution time of the task eligible for offloading by calculating the maxima and minima of the time and energy cost for remote execution. Whether offloading a task to the cloud server would save computation time and energy is determined using these maxima and minima values.

3.4.4 Statistical Regression Model

The decision of offloading a task is mainly dependent on the estimated time for the task to execute. The statistical regression model estimates the task execution time by modeling the dynamic usage behavior and environmental conditions. This model gives better estimation than the global averaging and the time and location based averaging techniques. The details are found in Section 4.3.

3.4.5 Device Clone

The clone of the Smartphone to be present at the cloud server is a pre-requisite for code offloading process to work. This device clone on the cloud server is capable of executing the task that is eligible for offloading from the mobile device. The device clone is loaded with the necessary application codes and data. When an Smartphone generates some new data that affect the execution of a task, the device clone should be updated with the new data through a suitable synchronization mechanism. On the other hand, when a new application is installed in a Smartphone, the cloud server should be able to obtain that new application clone either from the mobile device itself or from the Internet using other mechanisms.

3.4.6 Client Handler

Client handler in the cloud side is responsible for handling requests from Smartphones. It continuously listens on the port where a client makes a request and it first looks for the type of the request: *Control request* or *Task execution request.*

- **Control request**: The Smartphone periodically communicates with the cloud server by sending control request with a fake 10KB data [27]. This request is sent to the ExTrade server to obtain the connectivity information. When the client handler receives a control request from the client, it replies with similar data with one additional information about the current load of the cloud server. From this additional information, the client can calculate the speed factor. The significance of the speed factor is described in later sections.

- **Task execution request**: When a task is decided to be offloaded, the Smartphone sends a task execution request with application id and thread id to the cloud server. Upon receiving the task execution request, the client handler issues the execution of that task. After completion of the task, the client handler obtains the result of the execution and replies the client with the result. Along with the result, it sends the current server load information so that the profiler of the Smartphone can update the server speed factor.

3.5 Code Architecture

Our method requires the application developer to annotate the threads that are eligible for offloading (this is done in our implementation by setting a Boolean variable). This is a better choice than annotating the local threads since a thread annotated mistakenly can sacrifice the program correctness although in some scenarios it might require fewer annotations. When a thread is about to execute the system checks whether the thread is eligible for offloading. If it is, and it meets the computation and energy criteria, the thread is executed remotely; otherwise, the local execution is performed. The following constraints need to be considered when annotating a thread for offloading:

- Threads that update the user interface cannot be offloaded and

- Threads that access local device resources, that would hamper the remote execution of the thread, cannot be offloaded.

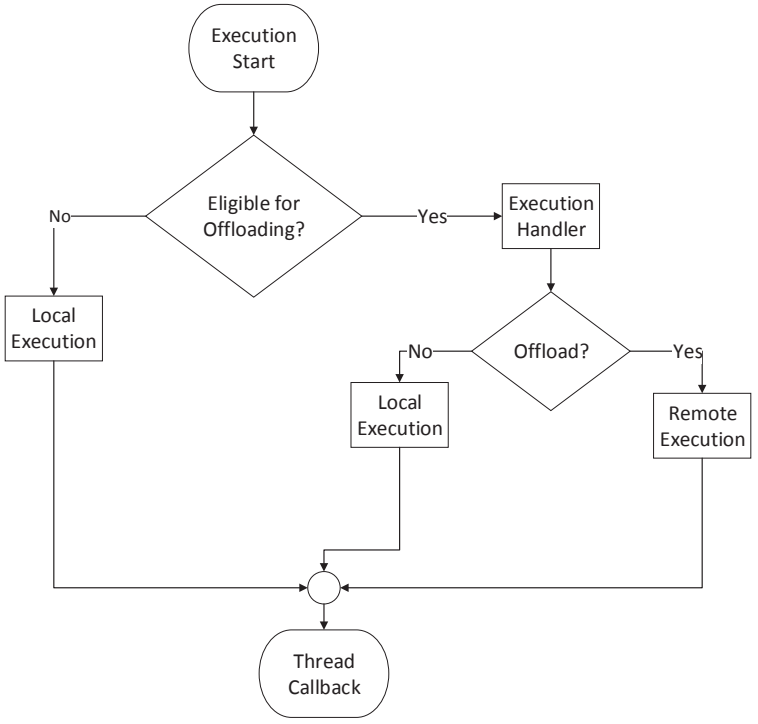


FIGURE 3.2: Work flow of ExTrade

Examples for the second case are reading data from phone audio input, GPS or taking picture in a facial recognition application, etc.

3.6 Work flow

Figure 3.2 shows the work flow of our proposed ExTrade system. At first, it is checked that whether a task is annotated for offloading or not. If the task is eligible for offloading, the control is handed over to the execution handler. As described in section 3.4 the execution handler then makes the decision whether the task should be offloaded or not based on the information obtained from the profiler, Lagrange solver and statistical regression model. After completion of the thread execution, either at local or remote, the callback is called and the normal subsequent execution is resumed.

3.7 Conclusion

In this chapter, we have presented the system architecture and the different components of our proposed ExTrade system. After that, we have described code architecture of our ExTrade system. In this chapter, we have given the work flow of our proposed work and how the decision handler makes the code offloading decision. In the next chapter, we describe our proposed decision making process and we also describe the process of estimating the task execution time of our ExTrade system.

Chapter 4

Decision Making Using Lagrange Multiplier

In this chapter, we discuss the decision making process of our ExTrade system. We mathematically formulate the problem of calculating the offloading cost and making the offloading decision. We also describe the process of tradeoff value calculation and the decision making technique in the tradeoff regions. An finally, we discuss the model of estimating the task execution time.

4.1 Introduction

When offloading code from mobile device to the cloud server it is important know whether the offloading process would save us the computation time and energy or it would cost us more. To offload a code only when it is beneficiary to the application user we need to define the conditions based on which the offloading decision would be made. In the following sections we describe the method of Lagrange multiplier to make the code offloading decision.

4.2 Decision Making Using Lagrange Multiplier

The control parameters for the code offloading decision making are the costs of computation time, energy requirement, speed factor of the cloud server, and data transmission time; and, they are not linearly related. For exmaple, the time cost saving, from a remote execution cost, doesn't always increase or decrease with speed factor of the cloud server. Rather, it shows different values depending on the instantaneous network performance. Therefore, the control variables are not linearly related to each other and thus solving the problem of decision making using a linear optimization solver is not applicable in this scenario. We opt for a nonlinear optimization method using Lagrange multipliers.

4.2.1 Mathematical Formulation of The Problem

For code offloading decision, we first calculate the time cost function, f_t, required for task execution at cloud server,

$$f_t(t_c) = t_c + \frac{D}{B} \tag{4.1}$$

where, t_c is the task execution time in the cloud server, D is the amount of data to be exchanged in between the Smartphone and the cloud and B is the available network bandwidth. The first part of the cost function is the actual time needed for the cloud server processor to execute the task and the second part is the time needed to send and receive data to and from the cloud server. Now, if the task execution time in the mobile device is t_m and the cloud server is S_f times faster than the mobile device, then the execution time of the task in the cloud server, t_c, is calculated as, $t_c = \frac{t_m}{S_f}$. Therefore, the above cost function can be rewritten as

$$f_t(t_m) = \frac{t_m}{S_f} + \frac{D}{B}. \tag{4.2}$$

We want to find the minima and maxima, using the Lagrange multiplier method, of the function defined below that is used to calculate the computation time saved

by offloading a task to the cloud server

$$F_t(t_m) = t_m - \frac{t_m}{S_f} - \frac{D}{B} \tag{4.3}$$

satisfying the following constraints:

$$f_t(t_m) < \frac{t_m}{S_f} + \frac{D}{B} < t_m \tag{4.4a}$$

$$\frac{t_m}{S_f} + \frac{D}{B} < D_{max} \tag{4.4b}$$

$$t_m > 0 \tag{4.4c}$$

$$D > 0 \tag{4.4d}$$

$$S_f \geq 1 \tag{4.4e}$$

Here, the first constraint (4.4a) states that the cost of task execution at mobile device should be greater than that of cloud server. This constraint ensures that offloading a task, whose execution time falls in between the minima and maxima range of the function, would save computation time. Constraint 4.4b is the maximum delay constraint that makes sure that the response time of an application, when a task is offloaded to the cloud server, does not exceed the maximum tolerable delay. The constraints (4.4c) and (4.4d) are nonnegativity constraints and constraint (4.4e) ensures that the cloud server is faster (or similar) than the mobile device. When solving for the minima and maxima for the cost function using Lagrange multiplier method, the (4.4a) and the (4.4b) constraints are integrated in the system of differential equations. Since, we have inequality constraints in our optimization problem, we use the Karush-Kuhn-Tucker (KKT) extension of the Lagrange multiplier method. Using the above mentioned constraints the KKT

equations can be written as:

$$\nabla F_t(t_m) - \lambda_1 \times \nabla(t_m - \frac{t_m}{S_f} - \frac{D}{B}) - \lambda_2 \times \nabla(D_{max} - \frac{t_m}{S_f} - \frac{D}{B}) = 0 \quad (4.5)$$

$$\lambda_1 \times (t_m - \frac{t_m}{S_f} - \frac{D}{B}) = 0 \quad (4.6)$$

$$\lambda_2 \times (D_{max} - \frac{t_m}{S_f} - \frac{D}{B}) = 0 \quad (4.7)$$

$$f_t(t_m) < \frac{t_m}{S_f} + \frac{D}{B} < t_m \quad (4.8)$$

$$\frac{t_m}{S_f} + \frac{D}{B} < D_{max} \quad (4.9)$$

$$\lambda_1, \lambda_2 \geq 0 \quad (4.10)$$

where, λ_1 and λ_2 are two constants known as Lagrange multipliers and ∇ is the derivative with respect to the independent variable t_m. Solving these quations would give us the maxima and minima, t^t_{max} and t^t_{min}, respectively, of the computation time saving function. Similarly, we could solve for the maxima and minima, t^e_{max} and t^e_{min}, of the energy saving function $F_e(t_m)$ as described below:

$$F_e(t_m) = p_m \times t_m - p_i \times \frac{t_m}{S_f} - p_t \times \frac{D}{B} \quad (4.11)$$

where, p_m is the average power consumed by the mobile device while doing computation, p_i is the average power consumed by the mobile device while being in the idle mode and p_t is average transmission and reception power of the mobile device. The process for decision making using these minima and maxima values is described in the following subsections.

4.2.2 Decision Making Policies

After calculating the maxima and minima of the time and energy cost functions, we introduce two types of decision making policies: *exact* and *tradeoff* code offloading.

- **Exact decision making policy**: In this policy, whether to offload a task to the cloud server or not can be decided deterministically. That is, it can be exactly stated that, offloading a task either would save both computation time and energy or neither. When offloading a task to the cloud server saves

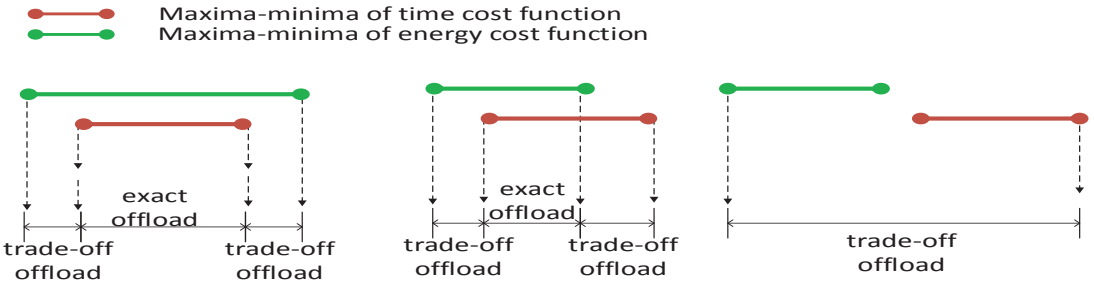
both computation time and energy, we always decide to offload; if it does not save any of those, we never offload.

- **Tradeoff decision making policy**: In this policy, it cannot be stated exactly that whether offloading a task to the cloud server would save both computation time and energy or not. On the contrary, in this policy, offloading a task results in saving one of the factors (*i.e.* computation or energy) while compromising other. Therefore, in this scenario, a *tradeoff metric* is required; if the value of the metric is acceptable, the task is offloaded; otherwise, the task is executed locally. The calculation of the *tradeoff metric* is described in detail in Section 4.4 and its acceptability is typically determined by the QoS requirements of the underlying applications. This policy is suited for various practical applications. For example, in a game application, it is enough to perform necessary computations to render 30 frames per second (fps) to meet the user QoS requirements. While most of these computations are heavy and require fairly large amount of power from the mobile device, today's Smartphones can perform up to rendering 60-120 fps. If offloading such a task in tradeoff region cuts down the response time to even half of the local response time, it would still not violate the user QoS requirements. Therefore, in such situations, code offloading could save significant amount of energy from the mobile device.

4.2.3 Decision Making Regions

For making decision using the results obtained from the previous subsection let us at first classify the tasks eligible for offloading into the following categories:

- **Compute intensive**: This type of tasks needs heavy computation with relatively less amount of data transfers. Solving N-Queen problem is a standard example for this class of tasks, where the number of queens to be placed in the chess board needs to be transferred to the ExTrade server for computing the result. While this requires a small amount of data transfers, the computation needed for solving the problem is fairly large. The decision of whether



(a) Full overlap (b) Partial overlap (c) No Overlap

FIGURE 4.1: Tradeoff and offload regions.

offloading these tasks would be beneficiary or not is mainly dependent on the amount of required computation.

- **Data intensive**: This type of tasks needs large amount of data transfers. For example, a face detection application might require to transfer a large image from mobile to the cloud. Generally, the tasks require the dynamic user inputs fall under this category, such as, matrix operation based computation tasks, sorting huge data records, etc. To offload this type of tasks to cloud server the network condition plays a very important role since a poor network would require more time to transfer the data and in turn would consume more transmission energy from the mobile device.

Fig. 4.1 shows the *offload* and *tradeoff* regions for different scenarios based on the maxima and minima values of the time and energy cost functions, discussed in the previous section. In Fig. 4.1(a), the maxima-minima range of one function is fully contained in another, *i.e.*, there is a full overlap, which indicates that the execution time belonging to that region would provide savings in both computation and energy by code offloading. The non overlapping region in the figure indicates that offloading the task would save either energy or computation time (in the figure it's energy), but not both, hence a tradeoff calculation is required to make the code offloading decision.

In Fig. 4.1(b), we observe partial overlapping of the time and energy costs. The tradeoff region on the left side indicates energy saving while compromising the computation time, and vice-versa on the right side. Similarly, Fig. 4.1(c) shows the scenario where only the case of improvement on either energy or computation

is possible because fully non-overlapping of the costs. In all the cases, the region outside the minima-maxima values is the region where offloading would neither save computation nor energy.

When an offloadable task is about to execute, the system will determine the expected time of execution of the task (to be described in Section 4.3) and compare it with the maxima and minima values of t_m of the cost functions. Depending on the value falling in the regions indicated by Fig. 4.1, the code offloading decision is made.

4.2.4 Advantage

The advantage of using lagrange multiplier for making the decision of offloading is that it is a non-linear optimization technique and it gets rid of the burden of solving a linear optimizer as it was mentioned in the earlier works. Solving a linear optimization problem is itself a compute intensive task. Performing a such a task in the mobile device is time consuming and it poses extra overhead for the mobile device. Also by using Lagrange multiplier we can determine the upper and lower bound on the amount of possible saving of the computation time and energy.

4.3 Estimating Task Execution Time

To make the decision of whether to offload a task or not it is important to determine the estimated execution time of the task, especially for the compute intensive tasks.

4.3.1 Statistical Regression

We use statistical regression technique [35] to estimate t_m. To describe the process, let us first model the execution time of the task, t_m, as a random variable, represented as follows,

$$t_m = m(X) + \epsilon, \tag{4.12}$$

where, $m(X)$ is the function of modeled parameters such as user inputs, current environmental situation, etc. and ϵ is residue factor due to the unmodeled parameters affecting the task execution time such as page faults, cache hit, etc. In this model, the goal of the task execution time prediction technique is to obtain estimates of $m(X)$ and ϵ given some parameter vector X.

To estimate the value of $m(X)$ and ϵ represented by $\overline{m}(X)$ and $\overline{\epsilon}$, respectively, our technique uses a set of previous observations of the execution time $\{t, X_i\}_{i=1}^{n}$. The statistical regression technique computes $\overline{m}(X)$ using the following equation,

$$\overline{m}(X) = \frac{\sum_{i=1}^{n} W_i(X) \times t_i}{\sum_{i=1}^{n} W_i(X)}, \tag{4.13}$$

where, $W_i(X)$ is the weighting function or kernel [38] and t_i is the execution time of the i^{th} observation. The weight function is designed in such a way that it assigns higher weights to the observations closer to the current situation and lower weights to the observations that are further away. In our implementation, we have used 7 closest previous observations in terms of the modeled parameters (user inputs and current CPU load) and given weight 0.5 for the first two observations, 0.3 for the next two, 0.2 for next two, and 0.1 for the seventh observation. For example, let's assume the last 7 observed execution time for a particular task closest to a particular input are [825, 855, 887, 814, 961, 868, 904]. Therefore, the estimated execution time of that particular task would be calculated as, $((825 \times 0.5) + (855 \times 0.5)/2) + ((887 \times 0.3) + (814 \times 0.3)/2) + ((961 \times 0.2) + (868 \times 0.2)/2) + (904 \times 0.1) = 948.45$

Units for all the data in the previous example is in milliseconds. Similarly, $\overline{\epsilon}$ is calculated using the statistical regression technique for the closest seven residue factors, as follows,

$$\overline{\epsilon} = \frac{\sum_{i=1}^{n} W_i(X) \times \epsilon_i}{\sum_{i=1}^{n} W_i(X)}. \tag{4.14}$$

The Fig. 4.2 shows how well the statistical regression based measurements for the

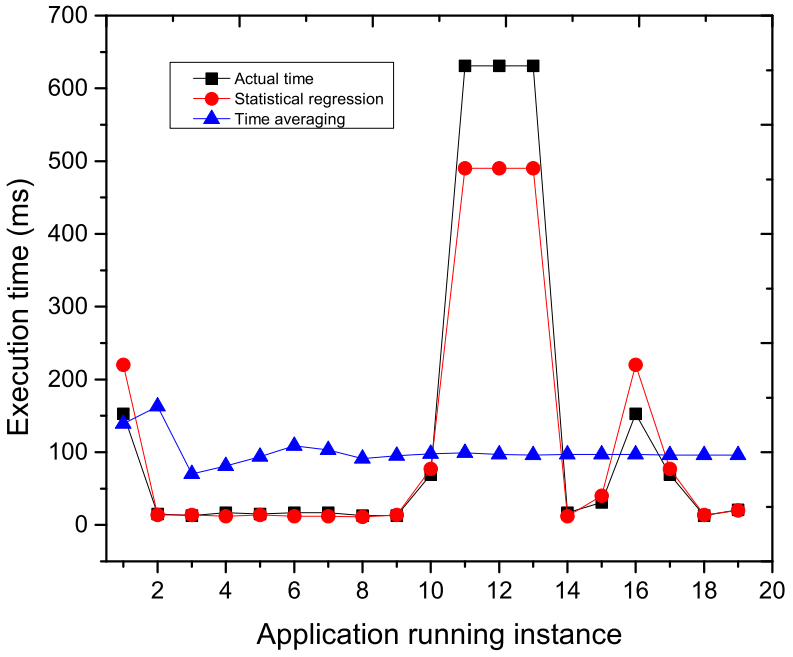


FIGURE 4.2: Prediction accuracy

estimated computation time of a task can predict the required amount of computation time compared to time-average based historical estimation techniques used in ThinkAir [29] and CADA [19]. The later techniques work well only when the task computation time does not vary highly. However, if the characteristic of the task is dynamic, for example, the user input to task is dynamically varying or the network condition is not stable, the time-average based prediction technique cannot capture the required time accurately. On the other hand, the statistical regression based model is quite capable of capturing the dynamic application usage behavior and the other environmental conditions when estimating the task execution time and thus, it makes the code offloading decisions more precise.

4.4 Tradeoff Analysis

Note that, as indicated in Fig. 4.1, the value of estimated computation time t_m might fall in the tradeoff region, where either computation time or energy could be saved while compromising the other. Given an acceptable amount of compromise is applicable, offloading a task might still be of benefit in the tradeoff region. Deciding what compromise is acceptable is dependant on the QoS requirement

of the application and the class of users using those applications. Therefore, we derive a *tradeoff metric* that helps to determine the decision. Let G_1 and G_2 represent the gains/loss achieved in time and energy, respectively, by offloading a task to the cloud, computed as follows,

$$G_1 = \frac{t_m - f_t(t_m)}{t_m + f_t(t_m)}, \tag{4.15}$$

$$G_2 = \frac{e_m - f_e(t_m)}{e_m + f_e(t_m)}, \tag{4.16}$$

where, e_m is the energy consumed by the device when executing the task locally and it is defined as, $e_m = p_m \times t_m$, and $f_e(t_m)$ is the energy cost of a task when it is offloaded to the cloud server; it is defined as, $f_e(t_m) = p_i \times \frac{t_m}{S_f} - p_t \times \frac{D}{B}$. Combining the above two factors, we get the *tradeoff metric* as follows,

$$G = \alpha \times G_1 + (1 - \alpha) \times G_2, \tag{4.17}$$

where, $0 \leq \alpha \leq 1$, which is the weight factor to tradeoff between the energy and computation savings. The value of α depends upon the remaining battery power of the mobile device. Therefore, the maximum weight is given to saving the computation time when the battery of the mobile device is full and the value of α is reduced as the remaining battery power of the mobile device decreases. Now, the location of task exectuion in the tradeoff region, denoted by D_{loc}, would be determined as follows

$$D_{loc} = \begin{cases} \text{remote}, & G \geq G_{th} \\ \\ \text{local}, & G < G_{th} \end{cases} \tag{4.18}$$

where, G_{th} is a predefined threshold which depends upon the QoS requirement of an application. Notice here that the value of α and G_{th} determine the acceptable compromise.

4.4.1 Determining the Weight Factor

Setting up a fixed value for α regardless of the situation (i.e. the remaining battery power of the mobile device) is not much of a practical use since the need of saving energy may change as the battery power of the mobile device goes down. Considering an adaptive policy for determining the value of alpha would be useful in practical scenario as this would give an opportunity to adapt the value of α as the residual energy of the mobile device changes with time and use. In this adaptive policy a high value for α (i.e. 0.8) is set when the amount of residual energy of the mobile device is high (i.e. 100 to 80 percent) and the value is gradually decreased as the remaining battery life goes down.

4.4.2 Conclusion

In this chapter, we have described the method of Lagrange multiplier to define the decision regions for making the code offloading decision. We have shown that there are two kinds of regions namely, the exact(or black-and-white) decision region and the tradeoff (or gray) decision region. In the exact decision regions one of the two conditions could arise. One condition ensures that both computation time and energy would be saved if the task is offloaded to the cloud server. In this condition, we always offload the task. The other condition implies that offloading the task to the cloud server would neither save energy nor computation time. In this condition, we never offload the task. In the tradeoff region only one of computation time and energy could be save while compromising the other. We have developed a tradeoff metric to make offloading decision in the tradeoff region. In the next chapter, we show the experimental result of our ExTrade system and compare the obtained result with some of the state-of-the-art techniques.

Chapter 5

Evaluation

In this chapter, we analyze the performance of our proposed ExTrade system based on different metrics such as, average computation time, average energy consumption, saving on computation time and prediction accuracy.

5.1 Experimental Setup

We have implemented our proposed system in a Samsung galaxy s2 tab. As the server we used a HP Pavilion g series commodity laptop with core 2 duo processor. The detailed device specification is given in table 5.1.

We considered two classes of application from the four classes as introduced in [19] such as, (i) Heavy Computation and Less Data transfer (HCLD) and (ii) Heavy Computation and Heavy Data transfer (HCHD). We developed a simple mobile

Table 5.1: Device specification

	Mobile	Server
CPU	Dual-core	Core 2 Duo
Clock Speed	1.2 GHz	2.1 GHz
Memory	1GB	4GB
Turbo Boost	No	No
Operating System	Android 4.0.2	Windows 7 64bit

application in each class and deployed to the device for performance analysis. The rest of the two classes that require low computation low data transfer and low computation high data transfer is not considered here because the applications that does not need heavy computation is less likely to be benefited from offloading code to the cloud server.

For class (i) application we need an application to carry out a huge amount of computation. For this purpose we implemented a N-Queen application that requires back tracking and consumes a significant amount of time from the mobile device for relatively less number of queens (i.e 5, 6 etc.). The data to be transferred for this application is fairly small.

For class (ii) we implement a face detection application that takes an image as the input and tries to find faces in that image if present. This application not only performs complex and heavy computation but also requires a large amount of data transfer if it is offloaded to the cloud server.

For calculating the amount of energy consumption, we used power tutor [39] an android application for monitoring energy usage for different applications that could estimate the energy usage with a fair amount of accuracy level.

5.2 Performance Metrics

We define the following performance metrics for analyzing the performance of the system.

- *Average execution time* - of a task is measured as the time difference between the instant at which the task starts its execution and the instant at which the result of the task is obtained. Execution time of an individual task is averaged over the total number of execution of the task.
- *Average energy consumption* - of a task is measured as the difference between the residual energy at the start of the execution of the task and at the end of the execution. Energy consumed by an individual task is averaged over the total number of execution of the task.

- *Prediction accuracy* - of a task is measured by comparing the expected task execution time predicted by the decision making mechanism with the actual time of the task execution. Suppose, the predicted execution time of a task is t_p, and the actual time of the task execution is t_a. Then, the prediction accuracy is calculated using the following equation,

$$Accuracy = (1 - \frac{|t_p - t_a|}{\max(t_p, t_a)}) \times 100\%. \tag{5.1}$$

  The accuracy of an individual task is averaged over the total number of task execution time.

- *Saving on computation time* - of a task is measured by subtracting the time needed for a task to execute on the remote server from the time the task needs to execute on the mobile device.

- *Operation overhead* - The operation overhead of a code offloading technique is measured as the ratio of total number of periodic request to the total number of task execution request sent to the cloud server over the course of application usage.

5.3 Result

5.3.1 Impact of Queen Size for N-Queens Application

In this section, we discuss the impact of queen size on N-Queens applications. The result of the experiment was obtained by running the application 10 times in each scenario and averaging.

In section 4.3 we introduced the technique of statistical regression for estimating the execution time of a task. One of the modeling parameters we considered is the user input for the task to estimate the execution time. Dynamic behavior of an application user is an important aspect to consider when estimating the execution time of the task. In earlier works like ThinkAir simple averaging of some previous execution history was taken as the estimation of the execution time of the task or in CADA the location and the time of the device was taken into consideration and

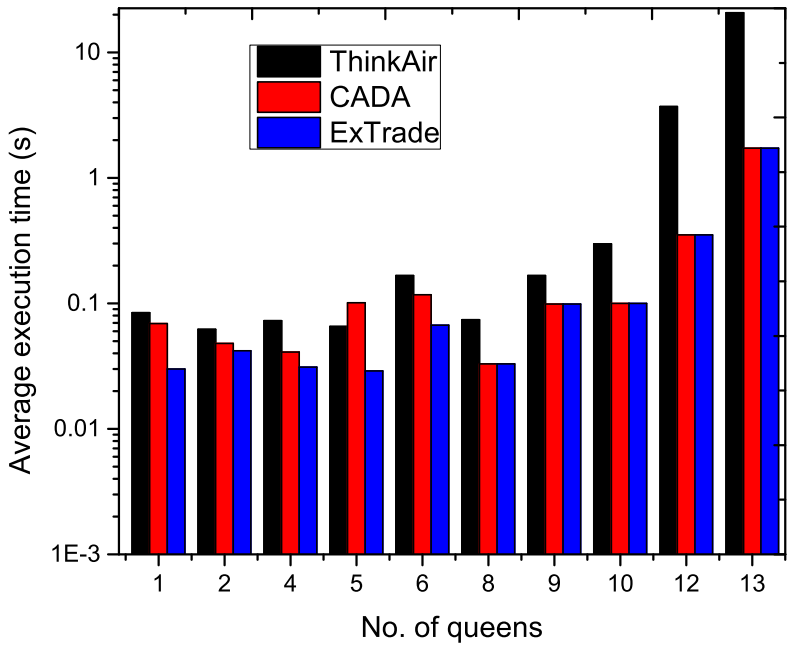


FIGURE 5.1: Average execution time of N-Queens application

previous observations of task execution time at the similar time and location was used to estimate the task execution time. Unlike these works ExTrade considers the user input as one of the modeling parameter for execution time estimation.

The experimental results, as shown in figure 5.1,5.2,5.3 and 5.4, shows the comparison of the execution time and energy consumption, prediction accuracy and saving on computation time of ThinkAir, ExTrade and CADA approach of the N-Queens application with respect to the queen size. As the application is running over a certain course of time the application user feeds input to the application dynamically. The impact of the dynamic input on the task execution time is significant. The offloading technique, that is highly dependent on the task execution time, must consider this dynamic behaviour of the application user for making correct decision. Figure 5.1 indicates that ExTrade saves more computation time than the other techniques for considering the dynamic behavior of the user.

Figure 5.2 shows the comparison of energy consumption of the N-Queens application. The mobile device consumes most of its energy when the mobile device is performing some computation. So by offloading a compute intensive task to the cloud server would result in less energy consumption on the mobile device. But

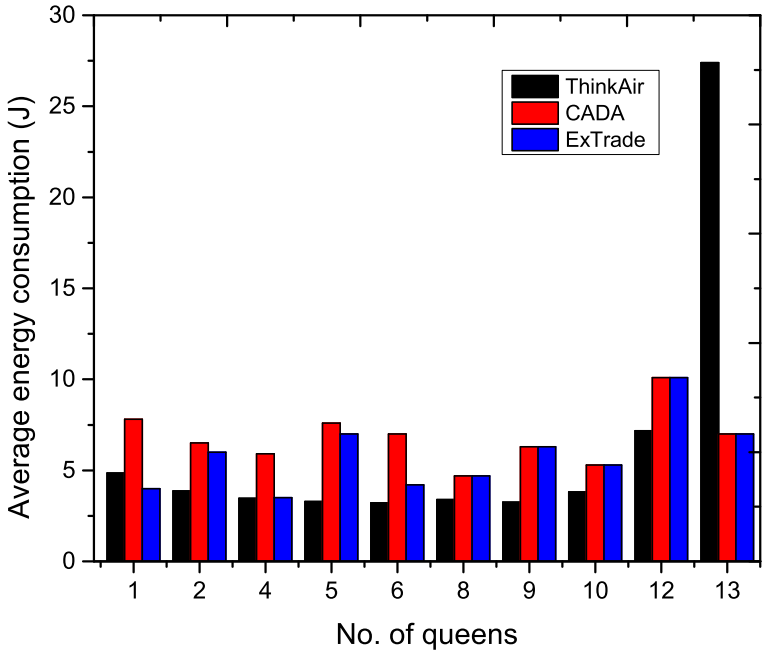


FIGURE 5.2: Average energy consumption of N-Queens application

offloading a task that is not so compute intensive as compared to the data transfer time needed may in reality increase the amount of energy consumed due to the transmission energy. As it is seen in the figure ExTrade generally saves more energy from the mobile device as it can more accurately predict the execution time of the task considering the dynamic behaviour of the user. But our in depth analysis of the experimental data shows that the CADA approach saves energy from the mobile device more than our technique in some the cases. This is due to some in accurate prediction of the CADA approach that offloads a task even when the offloading would not have been the right choice. But since the computation is offloaded, the computation energy needed from the mobile device is saved. However, this saving in the energy consumption comes with the cost poor response time from the application. Figure 5.3 shows prediction accuracy of the N-Queens applications with respect to the number of queens. The global averaging technique predicts the task execution time by averaging over a set of previous observation of the task regardless of other conditions. This technique fails to adapt to the dynamic user behaviour since it does not consider anything else than the previous observation history. The local averaging technique adopted by CADA approach takes the device location and the time of the day when the task is being executed

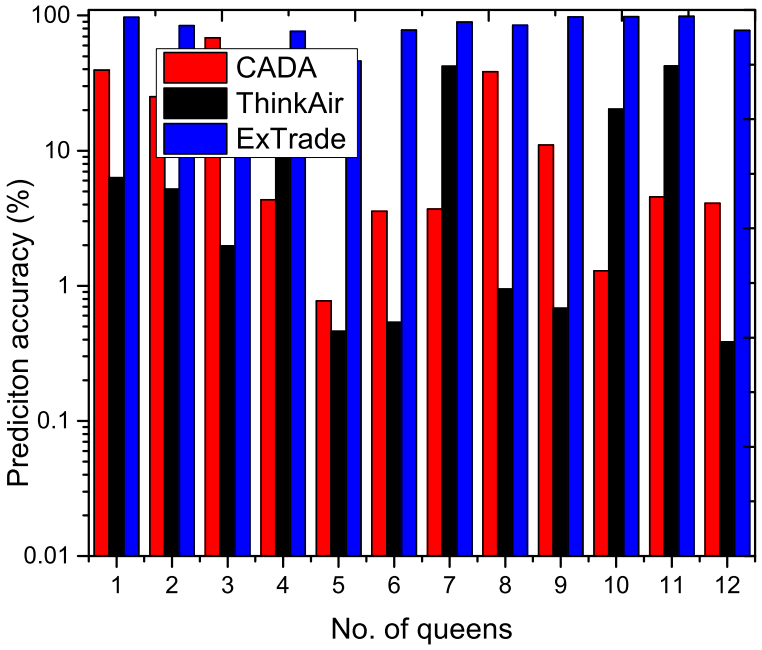


FIGURE 5.3: Prediction accuracy of N-Queens application

into consideration. While this technique can decide on the execution location of the task more correctly than the global averaging technique, their increased accuracy is only obtained when the user behaviour or other environmental conditions remain the same for a particular task on a particular time and location. But this technique cannot perform as good when the behaviour of application user does not remain similar. For a compute intensive application that depends on the user input modelling user behaviour allows for more accurate decision for offloading. Figure 5.4 shows the saving on computation time for the N-Queens application. As it is seen in the figure, in some of the cases the computation time saving for ExTrade and CADA approach is similar. This is due to the scenarios when both the approach make the correct code offloading decision. For example if a higher number of queen is input to the application, by choosing to offload the code to the cloud server both the approach could save similar amount of computation time. This generally happens for this application if a series of execution of the task is performed with higher number of queens. Then averaging the task execution time would give higher value as the estimated execution time and the code is likely to be offloaded.

5.3.2 Impact of Image Size for Face Detection Application

In this section we study the impact of image size of a face detection application on its computation time, energy consumption and prediction accuracy.

The experimental results, as shown in figure 5.5,5.6,5.7 and 5.8, shows the comparison of the execution time and energy consumption, prediction accuracy and saving on computation time of the face detection application with respect to the the image size dynamically input by the application user. Considering this dynamic input when making the code offloading decision is a must for making the offloading process effective. Figure 5.5 indicates that ExTrade saves more computation time than the other techniques for considering the dynamic behavior of the user.

Figure 5.6 shows the comparison of energy consumption of the face detection application. For data intensive applications like face detection transmission energy plays an important in the overall energy consumption of the application. The application will still consume most of its energy when it is in the computing state but since offloading such a task needs a high amount of data transfer, the transmission energy cannot be ignored. So the offloading mechanism must make

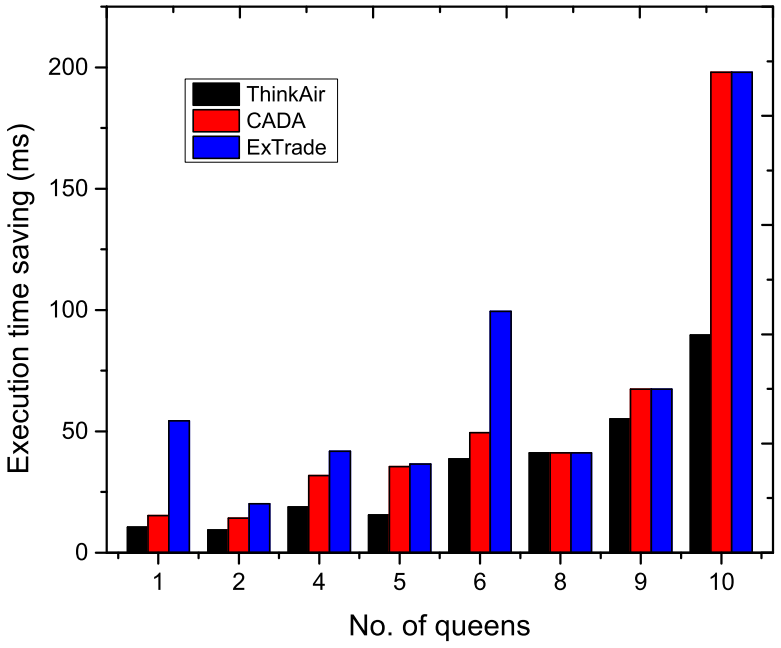


FIGURE 5.4: Average computation time saving of N-Queens application

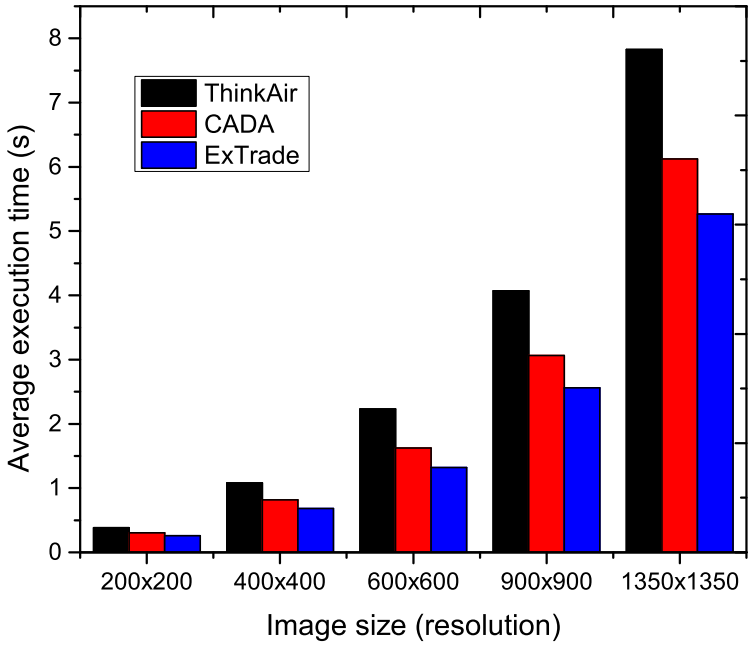


FIGURE 5.5: Average execution time of face detection application

the code offloading decision taking into consideration both the transmission time and the computation time. Since the data needed to be transferred is generally known prior to the actual transmission the prediction of the task execution time needs to be as accurate as possible because the offloading decision would then be

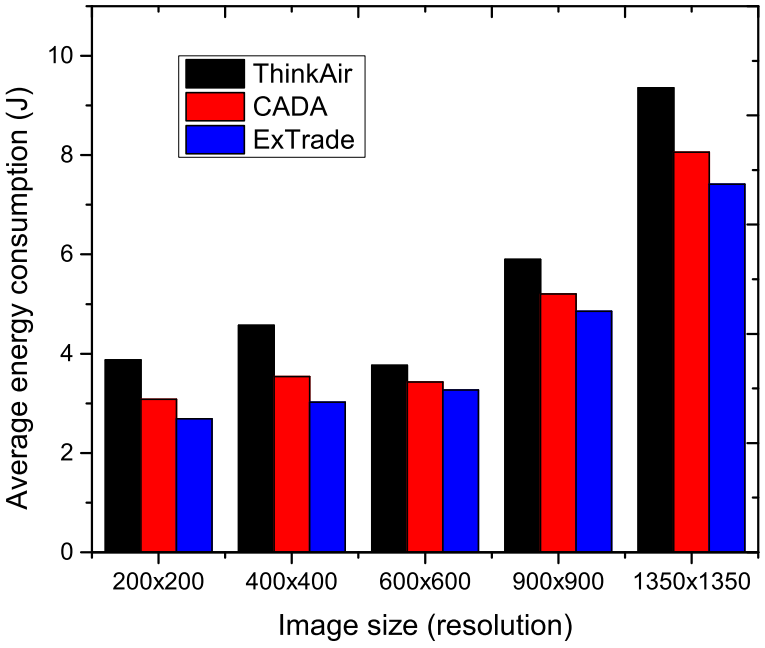


FIGURE 5.6: Average energy consumption of face detection application

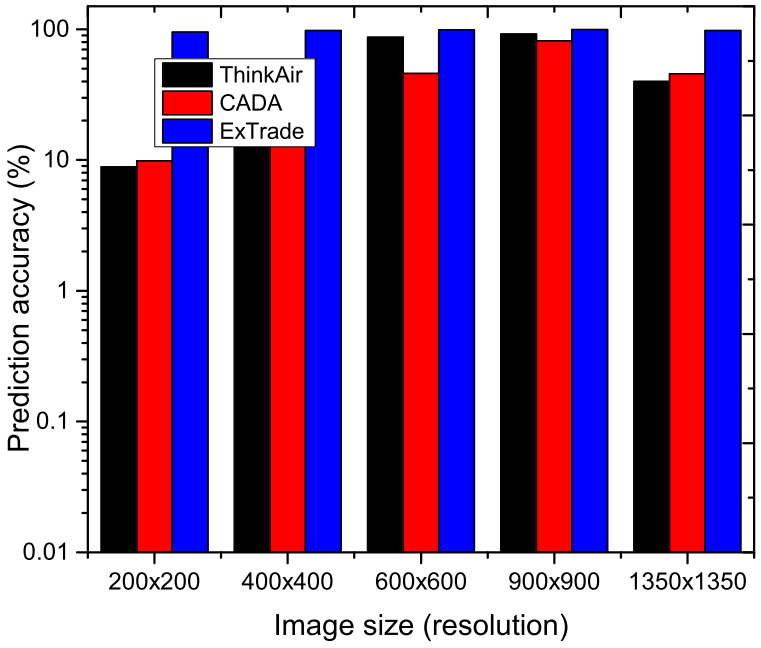


FIGURE 5.7: Prediction accuracy of face detection application

made by ensuring the improvement on the computation time would mitigate the needed transmission time. As it is seen in the figure ExTrade saves more energy from the mobile device as it can estimate the task execution time more accurately by considering dynamic user input.

Figure 5.7 shows prediction accuracy of the face detection applications with respect to image size. As stated earlier the global averaging technique predicts the task execution time by averaging over a set of previous observation of the task regardless of other conditions while the local averaging technique considers the device location and the time of the day when making the task execution time estimation. By modelling the dynamic user input ExTrade makes more accurate prediction of the task execution time.

Figure 5.8 shows the amount of computation time saving for the face detection application. Though on an average ExTrade saves more computation time, in many of the cases other approaches give similar performance. Our in depth analysis of the experimental data shows that when the application is run under similar condition (i.e. similar input and network condition), other averaging technique can predict the nature of the task quite accurately. For this reason the offloading

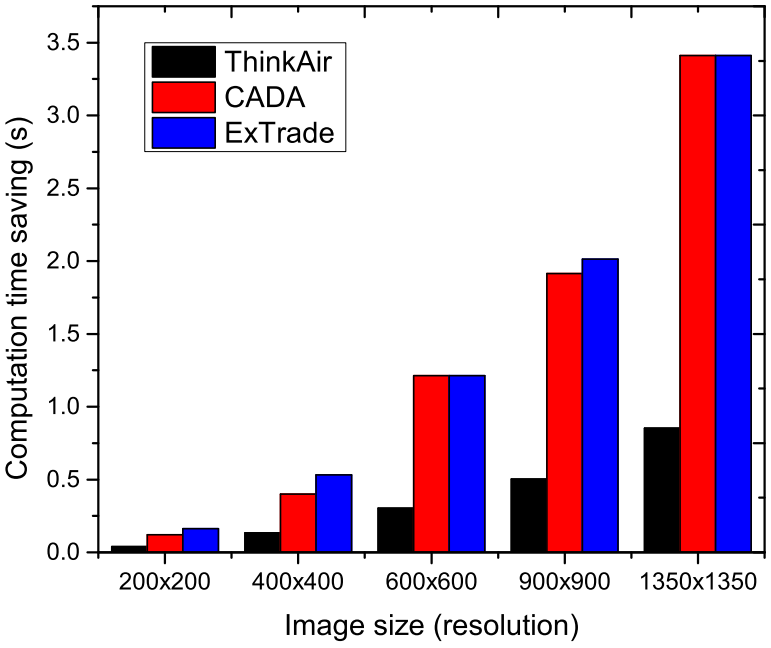


FIGURE 5.8: Average computation time saving of face detection application

decision made by other approach can become as accurate as the ExTrade approach. Hence, the saving on computation time can become similar for some scenarios.

5.3.3 Impact of Dynamic Network Condition

In this section, we discuss the impact of dynamic network condition on the performance of our offloading technique.

The dynamic behavior of the mobile network plays an important role when determining whether offloading a certain task from the mobile device to the cloud would be beneficiary to the application user or not. Poor network condition would result in more retransmission which adds to the application response time and energy consumption. In CADA the time and location of the device was considered when making the offloading decision assuming that the task execution condition would remain the same at a given location and at a given time of the day. But it would be a naive approach to consider the mobile network to remain the same or behave predictably at a certain location on everyday at a certain time. ExTrade profiler syncs with the cloud server periodically and keeps the updated information about

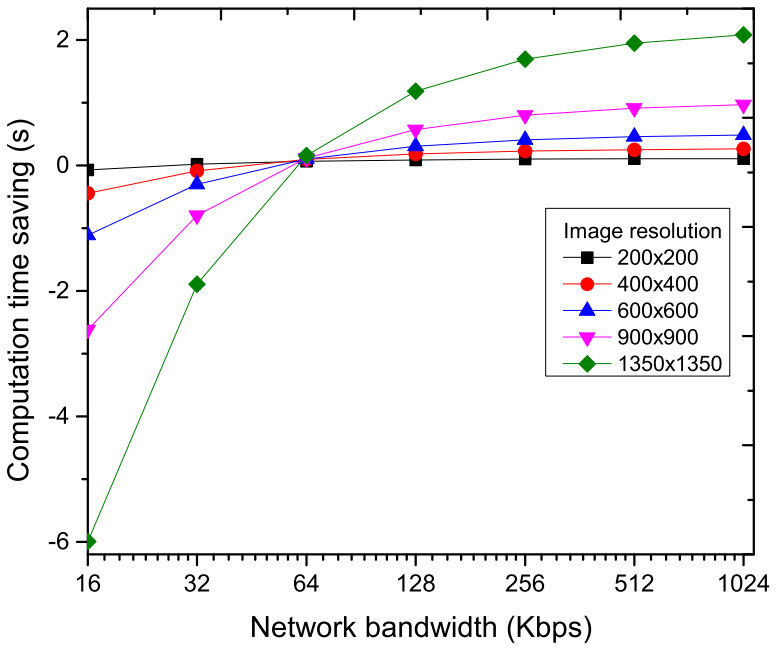


FIGURE 5.9: Impact of network bandwidth on face detection application

the network condition with help of which we could calculate the transfer time of remote execution request and make decision accordingly.

Figure 5.9 shows the impact of network bandwidth on computation time saving of the face detection application. Offloading a face detection task requires an image to be transferred to and from the cloud server thus the available network bandwidth plays a key part on offloading such tasks. We have studied the impact of network bandwidth on various image sizes and as shown in the figure, the impact of available network bandwidth becomes more critical as the image size increases because bigger image needs more time to transfer. In the figure, it is seen that the application with smaller images suffers from lower amount of penalty on computation time than the application with bigger images when the network condition in poor. On the other hand when the network condition is better application with larger images enjoys more computation time saving than the application with smaller images. This is due to the reason that the amount of saving on computation time is calculated by subtracting the time the task needs to execute remotely from the time the task need to execute locally. As a large image takes significantly more amount of time from the mobile device to process than a small image, the amount of saving is more. And when the network condition is

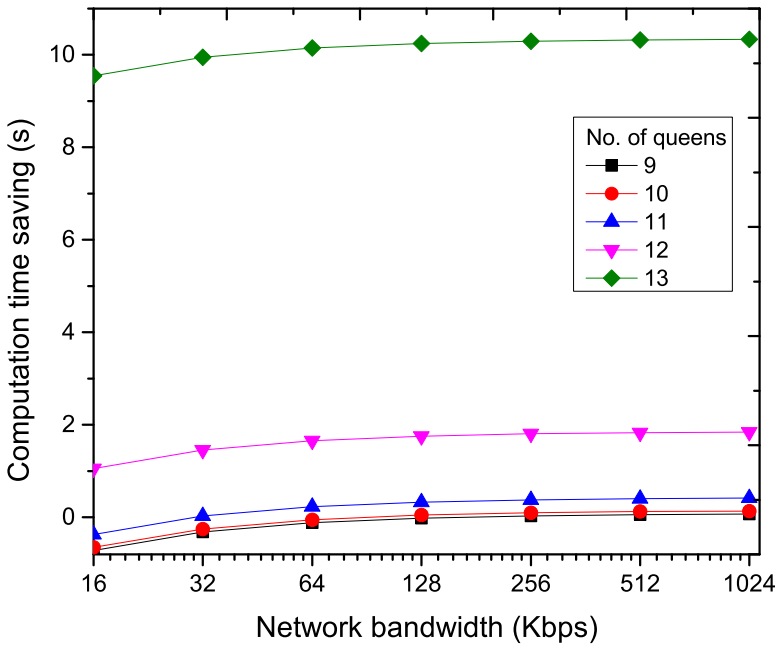


FIGURE 5.10: Impact of network bandwidth on N-Queens application

better the impact of transfer time on the overall application response time becomes less significant.

Figure 5.10 shows the impact of network bandwidth on computation time saving of the N-Queens application. As it is seen in the figure the impact of bandwidth on performance improvement of this application is not as much as compared to the previous application since the data needed to be transferred for this application is not of a high amount. As long as the network bandwidth is not so poor that the transfer time takes more than the computation time in the mobile device, this sort of compute intensive application can be benefited from code offloading. The figure also reveals that the computation time saving is higher for a larger number of queens than that of for a lower number of queens. While in reality, the time it takes to complete the calculation for a larger number of queens is greater. But since the computation saving is calculated by subtracting the remote computation time value from the local computation time value and local computation time gets even larger for a higher number of queens, the amount of time saved gets larger.

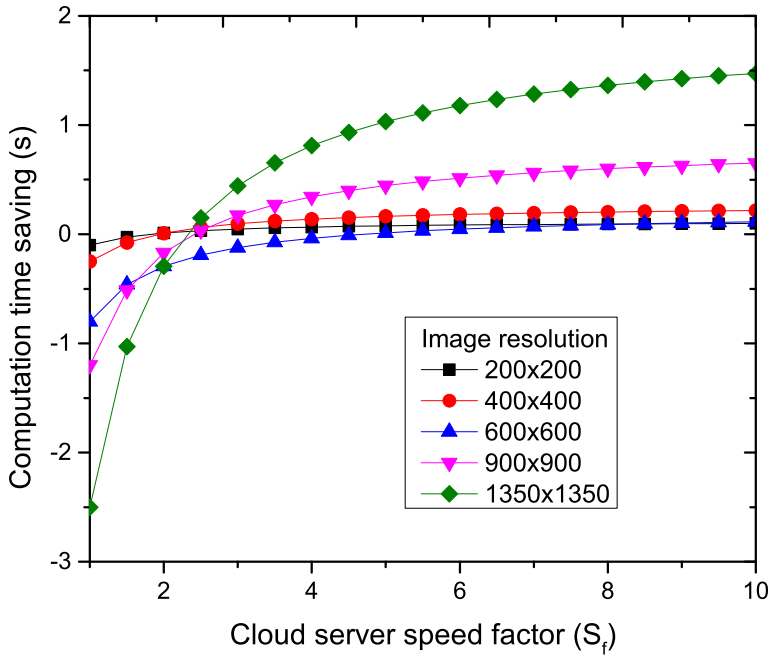


FIGURE 5.11: Impact of cloud server load on face detection application

5.3.4 Impact of Cloud Server Load

In this section we discuss the impact of cloud server load on saving computation time when offloading a code from the mobile device to the cloud server.

Dynamic load on the cloud server has a significant impact on the response time of an offloaded task. The less the load on the cloud server, indicated by higher values for the speed factor (S_f), the better the response time of the cloud server. ExTrade profiler keeps updated information on the cloud server load when it sends periodic packets to the cloud server and makes the offloading decision considering the current condition of the cloud server. The saving of the execution time is calculated by subtracting the value of the time needed for the task to be executed remotely from the value of the time needed for the task to be executed locally.

Figure 5.11 shows the impact of different cloud server load on computation time saving for the face detection application. Since a face detection application needs to send a picture to the cloud server when offloading the task for face detection, there is a significant amount of time spent on sending and receiving the data to and from the cloud server. So the speed factor needs to be of at least some minimum value that could mitigate the transfer time needed. As it is seen in

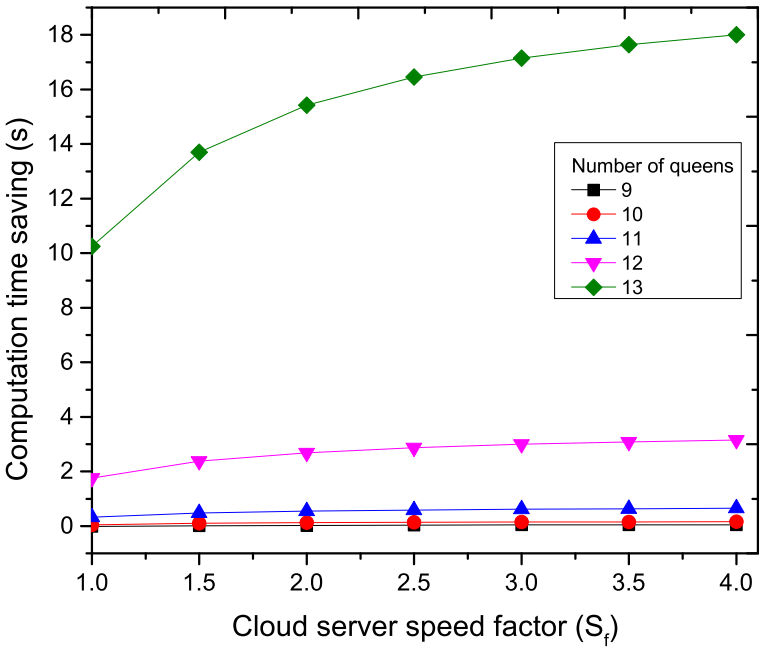


FIGURE 5.12: Impact of cloud server load on N-Queens application

the figure, in most of the cases this minimum value is 2. Our in depth look at the experimental data further reveals that the penalty acquired due to the data transfer time is higher when the image size is bigger and it is lower when the image size is smaller. This is due to the reason that bigger images need more time to be transferred and in those cases the speed factor needs to be of higher value (generally greater than 2) to save computation time. The figure also shows that the saving of the computation time gets larger as the image size gets bigger while looking at the experimental data reveals that the time needed by the cloud server to complete the task on a larger image gets bigger. But since mobile takes relatively more computation time than the cloud server as the image gets bigger in size, the amount of time saved gets larger.

Figure 5.12 shows the impact of cloud server load on computation time saving for N-Queens application. Since the application does not require a large amount of data exchange between the mobile device and cloud server the time needed to send request and receive result does not affect the overall performance of the application that much. On the other hand speed factor of the cloud server has a significant impact for this kind of applications. As it is seen in the figure the computation time saving amount is larger for large number of queens. But looking at the

experimental data reveals that the actual amount of time needed at the cloud server is greater for a large number of queens than that of for a small number of queens. But since the amount of saving on computation time is calculated by subtracting the time needed for the task to complete on remote server from the time needed for the task to complete on the mobile device and for larger number of queens the time a task need to complete on the mobile device is significantly larger than that of on the cloud server, the amount of computation time saving gets larger.

5.3.5 Operation Overhead

In this section, we compared the operation overhead of our proposed ExTrade system with the ThinkAir and CADA techniques. The small overhead is incurred to the system due to the periodic communication between the profiler of the smartphone and the cloud server for keeping the updated information about the available network bandwidth and the cloud server speed factor.

Our proposed ExTrade system opportunistically piggybacks the current information about the cloud server speed factor and available network bandwidth on the

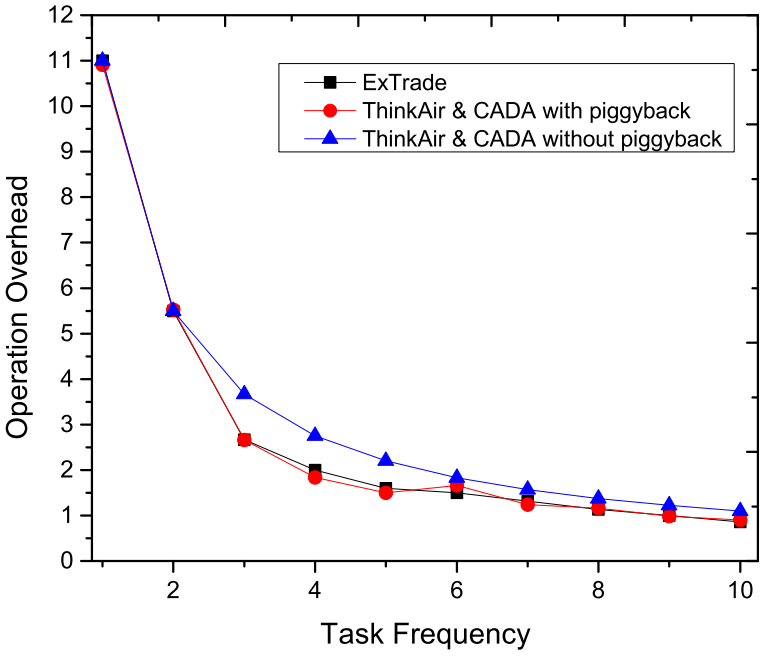


FIGURE 5.13: Operation Overhead

execution result of a remotely executed task. As it is not clearly stated in the work of ThinkAir and CADA that whether they use piggybacking mechanism or not; we implemented those techniques with piggybacking and without piggybacking for comparison. In x-axis we varied the frequency of the task execution over the course of time *i.e.*, we randomly varied how many times a task is going to execute within a predefined amount of time. Figure 5.13 shows, as the frequency of task execution grows the incurred system overhead is dramatically decreased for our proposed ExTrade system and other systems using piggybaking. The amount of overhead does not decrease as much for the techniques that do not use piggybacking. This is due to the reason that, with increased frequency of a task execution, more remote execution request is sent to the cloud server and the system with piggybacking gets the updated information more frequently and the overhead is reduced. On the other hand, the system without piggybacking does not exploit this opportunity to update the regarding information, therefore, the overhead does not decrease. Our proposed ExTrade system incurs somewhat similar amount of overhead as ThinkAir and CADA with piggybacking. Moreover, our in depth analysis of the experimental data reveals that in some execution events the overhead varies slightly between these techniques. This happens because of some incorrect code offloading decision made by ThinkAir and CADA techniques. In cases where a task is offloaded wrongly, it would result in less overhead and vice-versa.

5.4 Discussion

The experimental results show that the ExTrade can perform efficiently for offloading code from the mobile device to the cloud server under dynamic behaviour of the application user and environmental conditions. As a result ExTrade significantly improve the application response time and the energy consumption of the smartphone. It is expected to perform better when the characteristics of the application environment changes dynamically and does not stay consistent with the time and location of the application user. The applications that relies heavily on the user input and performs operations based on the user data are examples of such applications. However this technique might give performance similar to earlier works like ThinkAir and CADA when the application characteristic does

not change dynamically or the application user behaves in consistency with the time and location.

The statistical regression model used to estimate the task execution time that models the dynamic user inputs for estimation, predicts the execution time more accurately than the techniques used in earlier works. Due to this more accurate prediction mechanism the offloading decision, that depends highly on this estimation, can be made more effectively to improve application response time and energy consumption.

By considering the dynamic load on the cloud server to calculate the speed factor enables us to determine the cost value of offloading a task to the cloud server more effectively. This allowed us to accurately calculate the amount of time and energy that could be saved or would be lost by offloading the task. Thus we could make more accurate decision for offloading a task.

For data intensive tasks, profiling the current network condition allowed us to calculate the amount of transfer time needed to offload the task correctly. The impact of data transfer time is huge for a data intensive application. And a minimum level of speed factor needs to be ensured for an offloading process to be beneficiary for this type of applications. Together with the updated information of the cloud server load our work can calculate the amount of data transfer time and effective computation time on the cloud server. Using these information the offloading of a data intensive can be decided more efficiently.

However, the nice performances of our proposed ExTrade system do not come without cost. The extra overhead incurred in ExTrade system is due to the control information exchanges in between the Smartphone profiler and the cloud server for keeping updated information about the available network bandwidth and current computation load of the server. Furthermore, values of the tradeoff variables α and G_{th}, used in ExTrade, were determined through numerous simulation experiments. If we could build an analytical model for them, we could be able to dynamically obtain the optimal values to adapt to the changing computation environments. We left it as a future work.

Chapter 6

Conclusion

In this chapter, we summarize the research results presented in this thesis and state few directions for future works.

6.1 Discussion

We have proposed a non-linear optimization technique based on Lagrange Multiplier method to make the code offloading decision. This gets rid of the burden of solving a linear optimizer, as proposed in the earlier works, which requires heavy computation time and energy. We defined two types regions for making the offloading decision namely, exact decision making region and tradeoff decision making region. In exact decision making region or the black and white region, the decision of whether to offload a code or not could be made exactly as these regions either saves both computation time and energy or saves none. On the other hand the tradeoff decision making region or gray decision making region does not save both computation and energy. In the gray regions either computation or energy could be saved while compromising the other. We developed a tradeoff metric to make code offloading decision in the tradeoff regions.

We introduced a statistical regression based model for estimating the task execution time that predicts the execution time of a task by considering the dynamic behaviour of the environment and application user. Our experimental results show

that this model gives estimate of a task execution time more accurately than that of the earlier works. Due to its prediction accuracy the ExTrade technique makes more accurate offloading decision.

6.2 Future Scope

Use of multiple cloud server. In future we expect to take this work further and study the possibility of using multiple cloud servers to exploit the parallelization. When multiple cloud providers are available to the mobile device users, offloading the execution to a more capable cloud server could improve the performance of the application to a greater extent.

Parallel execution of task. If different parts of the task could be executed in parallel, delegating different part of the task to different server could be a possibility to improve the performance of the application even more.

Server with the best connectivity. Moreover, the cloud provider with the best network connectivity should be considered to cut down transfer delay and improve the response time of the application .In future we want to extend this work so that the capability of multiple cloud providers could be used as effectively as possible.

Appendix A

Submission Information

This paper is submitted to Elsevier's Future Generation Computer Systems titled: 'ExTrade: Exact and Trade-off Decision Making for Code Offloading in Mobile Cloud Computing', authors: Mahbub E Khoda, Md. Abdur Razzaque, Mohammad Mehedi Hassan, SK Md Mizanur Rahman, Atif Alamri.

Bibliography

[1] A. Smith, Smartphone ownership–2013 update, Pew Research Center: Washington DC.

[2] Survey reveals more smartphones activated each day than babies born (accessed on april 12, 2014), http://www.netgenie.net/blog/survey-reveals-more-smartphones-activated-each-day-than-babies-born.

[3] N. Fernando, S. W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems 29 (1) (2013) 84–106.

[4] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Cloudlets: bringing the cloud to the mobile user, in: Proceedings of the third ACM workshop on Mobile cloud computing and services, MCS '12, ACM, New York, NY, USA, 2012, pp. 29–36.

[5] M. Satyanarayanan, Fundamental challenges in mobile computing, in: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, ACM, 1996, pp. 1–7.

[6] D. Huang, et al., Mobile cloud computing, IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter 6 (10) (2011) 27–31.

[7] M. Mehta, I. Ajmera, R. Jondhale, Mobile cloud computing, International journal of Electronics.

[8] P. A. Cox, Mobile cloud computing, IBM developerWorks (2011) 1–10.

[9] D. Chalmers, M. Sloman, A survey of quality of service in mobile computing environments, Communications Surveys & Tutorials, IEEE 2 (2) (1999) 2–10.

[10] D. Ferreira, A. K. Dey, V. Kostakos, Understanding human-smartphone concerns: a study of battery life, in: Pervasive Computing, Springer, 2011, pp. 19–33.

[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[12] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation computer systems 25 (6) (2009) 599–616.

[13] X. Ma, Y. Zhao, L. Zhang, H. Wang, L. Peng, When mobile terminals meet the cloud: Computation offloading as the bridge, IEEE NETWORK 27 (5) (2013) 28–33.

[14] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, R. Buyya, Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges, IEEE Communications Surveys and Tutorials 99.

[15] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, B. Li, Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications, Wireless Communications, IEEE 20 (3).

[16] S. Abolfazli, Z. Sanaei, A. Gani, F. Xia, L. T. Yang, Rich mobile applications: Genesis, taxonomy, and open issues, Journal of Network and Computer Applications.

[17] `http://en.wikipedia.org/wiki/Mobile_cloud_computing`, accessed: May 07,2014.

[18] Y. E. Gelogo, H.-K. Kim, Mobile hybrid cloud computing issues and solutions.

[19] D. Shivarudrappa, M. Chen, S. Bharadwaj, Context-aware decision engine for mobile cloud offloading, in: Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2013, pp. 111–116.

[20] M. A. Iverson, F. Özgüner, L. Potter, Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment, IEEE Trans. Comput. 48 (12) (1999) 1374–1379.

[21] D. Shivarudrappa, M. Chen, S. Bharadwaj, Cofa: Automatic and dynamic code offload for android, Technical report published by University of Colorado, Boulder.

[22] `http://developer.android.com/guide/components/processes-and-threads.html`, accessed: April 21,2014.

[23] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, ACM SIGMETRICS Performance Evaluation Review 40 (4) (2013) 23–32.

[24] D. Kovachev, R. Klamma, Framework for computation offloading in mobile cloud computing., International Journal of Interactive Multimedia & Artificial Intelligence 1 (7).

[25] G. Huerta-Canepa, D. Lee, A virtual cloud computing provider for mobile devices, in: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, ACM, 2010, p. 6.

[26] Y. Ye, N. Jain, L. Xia, S. Joshi, I.-L. Yen, F. Bastani, K. L. Cureton, M. K. Bowler, A framework for qos and power management in a service cloud environment with mobile devices, in: Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on, IEEE, 2010, pp. 236–243.

[27] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 49–62.

[28] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the sixth conference on Computer systems, EuroSys '11, ACM, New York, NY, USA, 2011, pp. 301–314.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading., in: A. G. Greenberg, K. Sohraby (Eds.), INFOCOM, IEEE, 2012, pp. 945–953.

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

[31] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: a computation offloading framework for smartphones, in: Mobile Computing, Applications, and Services, Springer, 2012, pp. 59–79.

[32] S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, Y. Paek, Fast dynamic execution offloading for efficient mobile cloud computing, 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom) 0 (2013) 20–28.

[33] A. Wolbach, J. Harkes, S. Chellappa, M. Satyanarayanan, Transient customization of mobile computing infrastructure, in: Proceedings of the First Workshop on Virtualization in Mobile Computing, ACM, 2008, pp. 37–41.

[34] L. Luo, B. E. John, Predicting task execution time on handheld devices using the keystroke-level model, in: CHI'05 extended abstracts on Human factors in computing systems, ACM, 2005, pp. 1605–1608.

[35] W. Hardle, Applied nonparametric regression, Vol. 27, Cambridge Univ Press, 1990.

[36] J. Liu, K. Kumar, Y.-H. Lu, Tradeoff between energy savings and privacy protection in computation offloading, in: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '10, ACM, New York, NY, USA, 2010, pp. 213–218.

[37] Gartner says worldwide mobile device sales to end users reached 1.6 billion units in 2010; smartphone sales grew 72 percent in 2010 (accessed on april 11, 2014), `http://www.gartner.com/newsroom/id/1543014`.

[38] W. Hardle, Applied nonparametric regression (1994).

[39] Power tutor, `http://ziyang.eecs.umich.edu/projects/powertutor/`, accessed: April 21,2014.