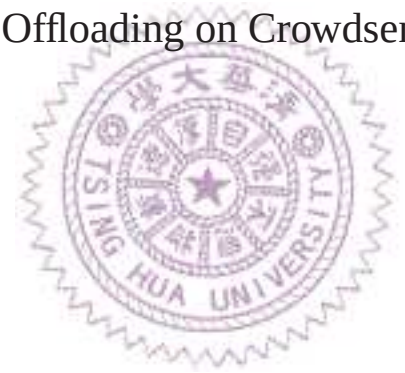
國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

群眾感測平台之行動雲端軟體卸載
Mobile Cloud Offloading on Crowdsensing Platforms



林庭毅
Ting-Yi Lin

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 103 年 12 月
December, 2014

國立清華大學
資訊工程研究所

碩士論文

群眾感測平台之行動雲端軟體卸載

林庭毅 撰

103
12

中文摘要

本論文專注於利用軟體卸載的技術減少行動裝置的耗電或事件分析演算法的處理時間，以提升群眾感測系統的系統效能。 事件分析演算法會經由使用者蒐集的感測器資料分析事件的發生與否，該分析可以透過卸載的技術卸載到雲端執行，以達到省電或提升速度的目的。由於軟體卸載的效益取決於許多因素，我們在本篇論文中提出了卸載決策演算法，結合當下的環境資訊，來決定是否將運算卸載到雲端。為了證明軟體卸載可實際用在現有的應用程式中，我們實作了一個工具幫助我們在沒有原始碼的情況下，分析其他第三方程式可以使用軟體卸載的部分，並修改該程式的安裝檔使其能夠使用軟體卸載。為了實際試驗軟體卸載對於群眾感測系統的影響，我們在本論文中實作了一個群眾感測雛形系統，並將軟體卸載的技術整合到該系統上進行測試。該雛型系統包含了(i)中控端、(ii)卸載伺服器及(iii)客戶端，並實作了2個事件分析演算法在系統中。

我們的實驗顯示了我們提出的卸載決策演算法可以達到80%的準確率，極大的避免了因決策錯誤產生的額外成本。同時，我們也透過自行開發的工具分析、修改第三方的程式，並經由實際的使用修改後的程式，證實了卸載技術的確能夠使用在現有的第三方程式中。最後，我們使用整合了軟體卸載的群眾感測系統進行實驗，證實軟體卸載的確能改進群眾感測系統的效能。

Abstract

We focus on utilizing offloading to reduce the energy consumption of the mobile devices or the execution time of event analysis algorithms, and improve the system performance of crowdsensing systems. Event analysis algorithms analyze the sensory data that are collected by mobile users to determine whether events happen or not. The overhead of event analysis algorithms can be offloaded to cloud servers to achieve energy efficiency or better performance. Since offloading benefits depend on many factors, such as network latency and computation capability, we propose an offloading decision algorithm, which considers the context information to decide whether to offload the computation to cloud servers. To show that offloading can be used in existing applications, we develop an APK analysis tool to analyze the third-party applications. Without the source code, the tool determines which part of an application can be offloaded and modify the application to be offload version. To evaluate the impact of crowdsensing system with offloading, we implement a crowdsensing prototype system and integrate offloading technique into the system. The prototype system involves (i) a broker, (ii) offload servers, and (iii) clients. We implement 2 event analysis algorithms in the system.

Our experiments show that our offloading decision algorithm improves the system performance through intelligently making offloading decisions, and the accuracy is up to 80%. The high accuracy significantly reduces the penalty of suboptimal offloading decision. Moreover, by utilizing the APK analysis tool, we analyze and modify third-party applications. The results of offloading the third-party applications show that offloading can be used in existing third-party applications. Last, we conduct experiments with the crowdsensing prototype system with offloading and show that offloading improves the performance of the crowdsensing system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Efficient Crowdsensing System With Offloading

Infrastructure sensors can be used to build a system providing abundant information, which is useful to citizens and governments. For example, monitoring the traffic condition and the air pollution level. These information make cities smarter and improve the quality of citizens' daily life. However, deploying the infrastructure sensors everywhere and maintaining the infrastructure sensors is costly to individuals, and is a heavy burden even to enterprises and governments. Although infrastructure sensors are cheaper and more powerful than before, deploying infrastructure sensors everywhere is expensive and takes a long time to achieve it. Not only the sensors, the wired/wireless connection and the plug-in power for sensors are also heavy expenditures.

Crowdsensing seems to be an alternative solution for infrastructure sensing. Nowadays, mobile devices are equipped with multiple sensors (e.g. GPS, accelerometer, etc.) and connectivities (e.g. Wifi interface, 3G/4G interface, etc.). This makes the mobile devices can be considered as mobile sensors. The popularity and mobility of mobile devices make the topic attracting to governments and people since the expenditure of deploying infrastructure sensors can be reduced if leverage the mobile devices properly.

The difference between crowdsensing and infrastructure sensing is that crowdsensing can be used in providing information that cannot be easily answered by computer. These information can be answered in crowdsensing system since humans are introduced to become part of the system. In this thesis, we use event to refer to the information that can be provided by the crowdsensing system. Events can be identified through event analysis algorithms, which analyze the collected sensory data to determine whether events happen or not.

Although crowdsensing is interesting and powerful, it has several issues that should be addressed in order to deploy it in real world. In this thesis, we address on improving

the performance of event analysis algorithms through offloading. The event analysis algorithms may lead to heavy computation, and take a long time to execute on mobile devices. It is not energy efficient to execute on mobile devices. This is similar to offloading, and we try to improve crowdsensing systems through offloading heavy computation to cloud servers to reduce the energy consumption of mobile devices or the processing time of event analysis algorithms.

1.2 Offloading Applications To Cloud

Cloud computing is getting increasingly popular for several reasons, such as lower maintenance cost, more elastic resource allocation, and easier access. Mobile devices, such as smartphones and tablets, are becoming ubiquitous because modern mobile devices are more and more powerful and 3G/4G cellular networks provide fast mobile Internet access. Nonetheless, mobile devices still have stringent resource constraints on, e.g., CPU power, memory size, storage space, and battery lifetime, compared to laptops and desktops. Among these constraints, the battery lifetime is probably the most critical one for mobile users [1], and the response time also imposes negative impact on the user experience.

Mobile applications may cope with the constraints by *sacrificing* user experience, e.g., rendering videos at lower quality and returning the search results after longer response time. A better solution is to offload computationally-intensive processing over the wireless networks to the cloud servers in order to reduce the energy consumption or the response time. This is referred to as *mobile cloud offloading* [9, 11, 17]. Mobile cloud offloading allows the mobile applications to achieve better user-experience, and offers the cloud service providers more business opportunities.

In this thesis, we study the *decision engine* in mobile cloud offloading systems, which decides whether to offload a given method to the cloud servers. A naive approach is to *always* offload, which may lead to higher energy consumption and longer response time if the method is not computationally-intensive and the data to be transferred is large. We propose a better, context-aware decision engine, which leverages on user context and historical measurements to make offloading decision for minimizing energy consumption, response time, or other optimization criteria. To our best knowledge, context-aware mobile cloud offloading has not been rigorously studied. While our proposed solution can be integrated in various cloud offloading systems, we have built a proof-of-concept prototype on top of ThinkAir[1] [17]. Our preliminary experimental results from several real Android phone users show that the proposed decision engine: (i) achieves more than 80% predic-

[1]We thank ThinkAir's authors for sharing their source code.

tion accuracy, (ii) improves the overall performance in energy consumption or response time, and (iii) incurs very low overhead.

1.3 Contributions

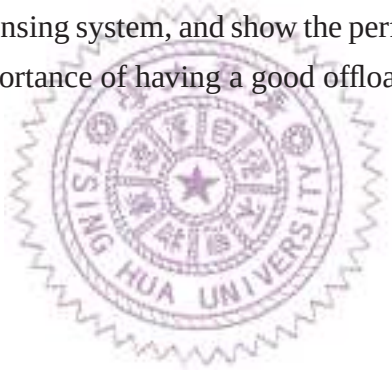
The contributions in this thesis are

1. Implement a real crowdsensing prototype system. The offloading technique is integrated in the system and can be deployed in real machine [20].

2. Propose a context-aware decision algorithm. This algorithm determines whether to offload the computations to cloud servers with high accuracy [21].

3. Analyze and modify existing applications to show that offloading can be used in existing applications.

4. We deploy the crowdsensing system, and show the performance improvement through offloading and the importance of having a good offloading decision algorithm.



Chapter 2

Related Work

2.1 Offloading

Mobile cloud offloading has been considered in the literature [9, 11, 17]. MAUI [11] is an offloading system designed for Windows phones. It uses Microsoft .NET to identify the methods that can be offloaded and the states that needed to be transferred during offloading. MAUI continuously collects essential data, e.g., energy consumption, CPU utilization, and network conditions, at runtime. It uses the collected data to make the decision on whether to offload for saving energy of mobile devices. In particular, MAUI builds a call graph and solves it as a 0-1 integer linear programming problem. ThinkAir [17] enables method-level offloading system on Android phones. Similar to MAUI, developers should add notations to the methods they want to offload. ThinkAir logs the energy consumption, execution time, and network conditions, which are used to decide whether to offload. ThinkAir makes offloading decisions based on user preferences and the logs. Clonecloud [9] is an offloading system for Android. Different from MAUI [11] and ThinkAir [17], it directly works with application binaries. The code partitioning is completed by a static analyzer which builds a control-flow graph and identifies the possible partition choices. The dynamic analyzer uses random input datasets to generate multiple execution logs, which are used to solve an integer linear programming problem. Kwon and Tilevich [18] use checkpoint to support offloading and minimize the states to be transfered. They implement a code enhancer to identify all methods annotated with a special tag and insert checkpoints in them to build the client- and server-side classes. Mobile applications are always offloaded as long as network is available. Since it doesn't check whether offloading can save energy, under some circumstances the energy consumption may be higher.

Several energy (or power) consumption models have been proposed for mobile devices [7, 13, 22, 24, 26, 30]. Roughly speaking, the energy consumption can be classified

into three components: (i) computation, (ii) communications, and (iii) display and others. In this paper, we focus more on communication energy consumption because it is directly affected by different contexts, while other components are rather static. Several energy models are base on the observation that different states of a network interface impose diverse power consumption levels [7, 24, 30]. Compared to [24, 30], Balasubramanian et al. [7] consider two extra network-relevant states: ramp and tail energy. Ramp energy is the energy consumed when the interface switches to the high power level before data transfer and tail energy is the energy consumed when the interface keeps in a high power level after data transfer for a system specified duration. Dong and Zhong [13] show the dependencies between energy models and mobile hardware, and develop a self-constructive and adaptive power modeling method. This model does not take different network conditions into consideration. The work in [22,26] takes network conditions into consideration. In [26], network conditions affect the transfer successful ratio. It is assumed that a failed transfer will be retransmitted under the same network condition. In [22], the communication energy is a function of the signal strength. Neither [26] nor [22] consider the implications of network congestion level.

2.2 Crowdsensing

Due to the advance of technology, smartphones are equipped with more and more sensors, such as GPS, accelerometer, and gyroscope. This motivates researchers to leverage the capability of smartphones to provide sensory data instead of deploying infrastructure sensors everywhere. Such a paradigm called *mobile crowdsensing* [14]. To solve specific problems, several crowdsensing applications are proposed in [8, 10, 16, 28, 31]. Chon et al. [8] aim to automatically identify places by crowdsensing. Through combining the radio finger print of WiFi access point and image processing, their framework classifies the places into gym, restaurant, etc. To improve image-based location reliability, Talasila et al. [28] introduce human validation and sensory data that are attached to the photos to determine the locations. Hasenfratz et al. [16] propose system that uses sensor and GPS to provide air quality map to users. Zhou et al. [31] propose a system to predict the bus arrival time. Through processing the sensory data that are collected by users, such as microphone and cell tower ID, the system identifies whether users are on buses and bus routes, and predicts the time for the bus to arrive at each bus stop. Some works use pre-install specific sensors, monitors in buildings, or event data recorders in cars to help sensing like Coric and Gruteser [10] and Lan et al. [19]. Coric and Gruteser [10] aim to provide on-street legal parking slot map to users. They assume the parking map is accessible in advance, and use the sensors and GPS installed on the cars to determine the

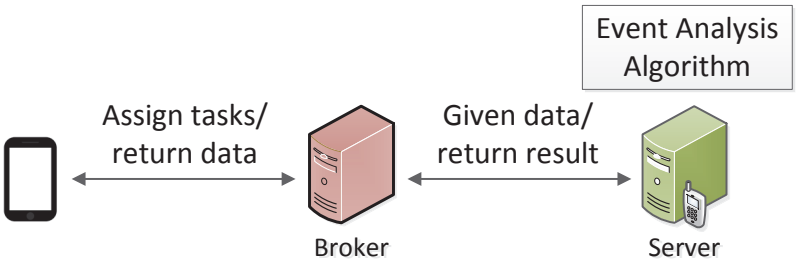


Figure 2.1: Process the sensory data on broker/server.

parking slots are occupied or not. The system shows the available parking slots on the map to avoid users wasting time on finding parking slots.

Besides the applications, other issues in crowdsensing systems are studied in [6, 19, 25, 27]. Since incentive is also an important issue in crowdsensing systems, Lan et al. [19] propose a framework which has incentive mechanism to motivate users to provide sensory data. Users who provide sensory data or share their connectivities can earn virtual credits. To achieve efficient data transfer in crowdsensing systems, Sherchan et al. [27] propose a framework that reduces energy and bandwidth consumption for sensory data collection and data transfer. Medusa [25] and Agarwal et al. [6] provide general crowdsensing systems to support various applications. Medusa [25] is a crowdsensing framework which aim to relieve the burden of developing new applications. Developers only have to express the sensing tasks through their own scripting languages, and Medusa automatically deploys the tasks to mobile clients to instruct users to perform the tasks. Agarwal et al. [6] provide a framework to ease the development of new applications. To avoid duplicate and in-efficient sensing, they also provide sensor schedule to make users perform the tasks efficiently.

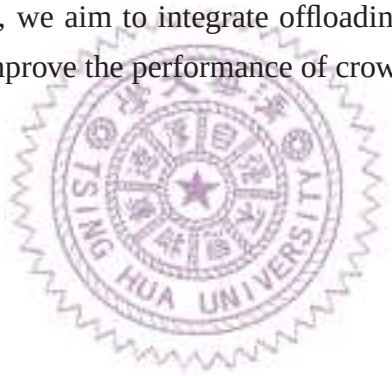
2.3 Challenge in Large-scale Crowdsensing Systems

Although the proposed crowdsensing systems are interesting and useful, to be deployed in real world, there are several challenges. These challenges should be addressed in order to make the proposed crowdsensing architecture deployable and better. In this thesis, we address on the *event analysis offloading problem*, which is one of the major challenges in crowdsensing systems. Since the events are detected by event analysis algorithms, there should be a host to take on the computations. As shown in Fig. 2.1, the most intuitive approach of performing event analysis algorithms is sending all the collected sensory data to a specific server and perform the event analysis algorithms on it. In fact, most of the crowdsensing systems follow this approach. However, this may lead to heavy

network overhead on mobile devices and severs in large scale systems. Therefore, this will consume larger amount of energy of transferring the sensory data on mobile devices. An alternative solution is to pre-process the sensory data before transferring to servers or even performing the event analysis algorithms on mobile devices. With this approach, we can significantly relieve the communication overhead of mobile devices and servers.

This becomes a trade-off between transmission and computation, which is similar to offloading. Therefore, we believe offloading can improve the performance of crowdsensing systems. By carefully determine whether execute event analysis algorithms on servers, energy consumption and processing time can be reduced in crowdsensing systems. For mobile users, they can also reduce energy and time through considering the trade-off of computation and transmission.

The biggest difference between our work and other existing crowdsensing systems is that they do not consider the execution place of event analysis algorithms. We consider the trade-off of transmission and computation to maximize the energy and time efficiency of the system. In this thesis, we aim to integrate offloading into crowdsensing systems and leverage offloading to improve the performance of crowdsensing systems.



Chapter 3

Offloading Applications To Cloud

3.1 Architecture

3.1.1 Overview

Fig. 3.1 shows the common mobile cloud offloading architecture. The resource-constrained mobile device is connect to the resourceful cloud server via the Internet. The mobile device and cloud server run a cloud offloading system, which is responsible to offloading a method execution from the mobile device to the cloud server, and then retrieve the return value from the cloud server back to the mobile devices. Various cloud offloading systems [9, 11, 17] proposed in the literature may be used here. Our work concentrate on the development of the decision engine. Before the mobile device executes a method, the cloud offloading system checks with the decision engine to see whether to offload that method. The decision engine should only instruct the cloud offloading system to offload the method if offloading the method results in better performance. Possible performance metric include energy consumption and response time, and is chosen by each mobile user. In this thesis, we design and evaluate a context-aware decision engine, which consists of four components: context-aware decision algorithm, context profiler, energy model, and context database. We detail each of them in following sections. We notice that exist mobile cloud offload systems [9, 11, 17] may also implement some decision heuristics, but to the best of our knowledge, they are not context-aware. In Sec. 3.4.1, We will show the importance of context-aware decision engines.

3.1.2 Context-aware Decision Algorithm

The decision algorithm is responsible for determining whether to offload. It considers the contexts that consists of: (i) time-of-day and (ii) location to decide local or cloud execution. We will discuss the context-aware decision algorithm in more details in Sec. 3.3.

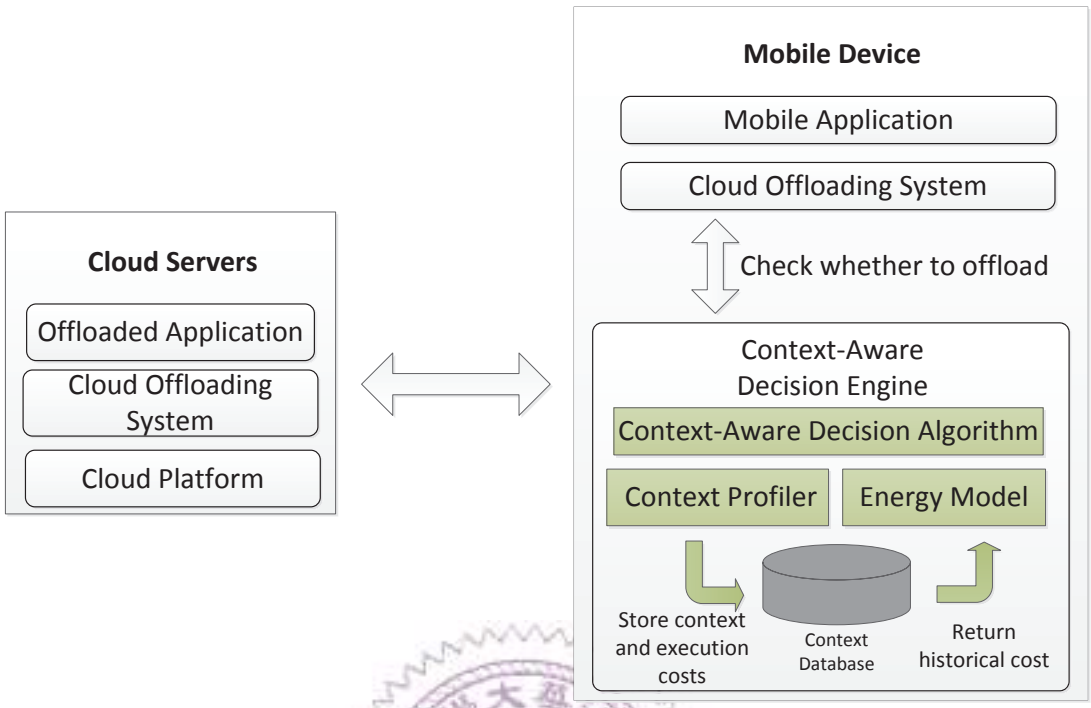


Figure 3.1: Mobile cloud offloading architecture.

3.1.3 Energy Model

The energy model estimates the execution energy consumption for the mobile device. It takes signal strength and transmission time to be input and output the energy consumption. Energy model is necessary in offloading system. Without energy model, we cannot understand the energy consumption of local execution and offloading, and we cannot determine whether offload or not. We discuss more details of our energy model in Sec. 3.2.

3.1.4 Profiler

The context profiler is a background service that collects various contexts required for making the offloading decisions. We consider the following four contexts:

- Signal strength: the profiler periodically collects the latest signal strength of WiFi and 3G networks.
- Transmission time: the profiler periodically transmits some dummy data to/from the cloud server to measure the end-to-end network throughput and hence get the transmission time for energy estimation.

- Time-of-day: the profiler partitions each day into 48 half-an-hour time slots. The time slots of the offloading opportunities are recorded.

- Location: the profiler logs the geographical location using GPS once every half an hour.

These contexts are saved in a database, and then used by the proposed context-aware decision algorithm and energy model.

3.1.5 Database

Once a method is executed, the costs including the processing time, energy consumption, and starting time slot are recorded in the database. The energy consumption is estimated using an energy model, which is based on the network signal strength and throughput. The derivation of our energy model is detailed in the next section. Since we only use the current signal strength and throughput, these contexts are not stored in the database. The CADA algorithm uses the location and time-of-day as the index and the database returns the average performance on the cloud and on the mobile device. If there is no previous record at the combination of location and time-of-day, the database returns null to the CADA algorithm.

3.2 Problem Statement

In this section, we discuss two significant problems that should be addressed in order to make applications benefit from offloading. The problems are (i) energy measurement and (ii) application modification without source code.

3.2.1 Energy Measurement

Energy measurement is necessary in order to understand the cost of local execution and offloading. We aim to leverage offloading to reduce the energy consumption or execution time of mobile devices. Therefore, we need an energy model to measure the energy consumption and use the result to determine whether to offload. The more accurate of the energy measurement, the more accurate decision that we can make. Also, the penalty of wrong decision can be high, hence, the accuracy of the energy model should be in a tolerant rate. In this section, we summarize a power profiling tool in the literature, and enhance it by incorporating user contexts for higher accuracy.

Table 3.1: Fitting Error of Different Mapping Functions

MSE	Linear	Quadratic	Cubic
Cellular	5.08×10^{-4}	2.20×10^{-4}	1.77×10^{-4}
WiFi	6.52×10^{-5}	4.35×10^{-5}	4.25×10^{-5}

Limitations of Existing Energy Models

PowerTutor [5] is an open-source power profiling tool for mobile devices. Its underlying power model is written as:

$$P_{total} = P_{cpu} + P_{comm} + P_{display} + P_{other}. \tag{3.1}$$

In Eq. (3.1), the total power is the sum of the component-wise power consumption. There are four components: P_{cpu}, P_{comm}, $P_{display}$ and P_{other} for computation, communications, display, and others. P_{comm} is further divided into the power consumption of WiFi and cellular networks:

$$P_{comm} = P_{WiFi} + P_{Cell}. \tag{3.2}$$

The Wifi and 3G power consumption is written as:

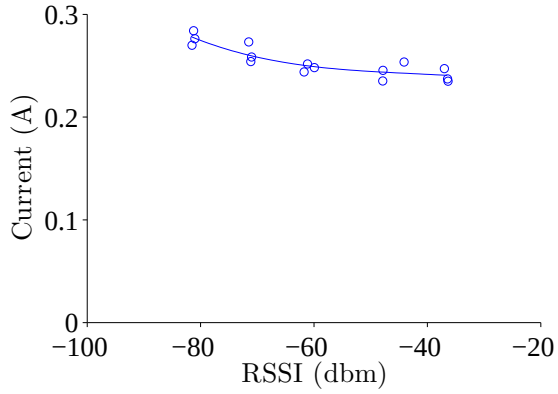
$$P_{WiFi/Cell} = P_{idle} \times \beta_{idle} + P_{trans} \times \beta_{trans}, \tag{3.3}$$

where P_{idle} and P_{trans} are the power consumption when the network interface is idle and transferring, and β_{idle} and β_{trans} denote the fraction of time the network interface is in different states. More details of the energy model can be found in [30].

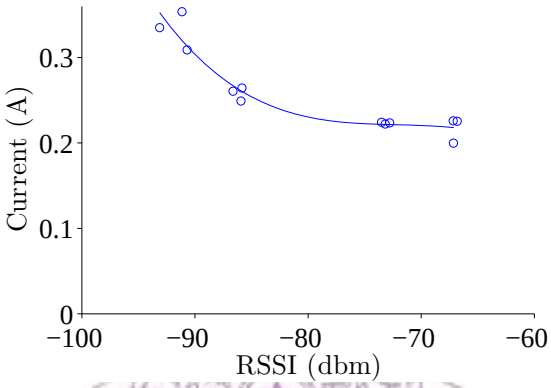
The communication power consumption depends on the signal strength and network congestion level. However, the communication power model in Eq. (3.3) does not take such network dynamics into considerations, and may lead to less accurate power estimation. Therefore, we develop a context-aware power model in the next section.

A Context-Aware Communication Energy Model

We propose a context-aware energy model, which is a function of the transmission power and congestion level. The communication energy consumption depends on: (i) the transmission power as it determines the instantaneous power consumption and (ii) the congestion level as it determines the data transfer time. However, most mobile OS's, including Android, do not provide APIs for polling the transmission power and congestion level. Therefore, we employ the two contexts in our energy model: the Received Signal Strength Indication (RSSI) and the throughput, because the transmission power is a function of the RSSI value, and the congestion level is a function of the network throughput. The RSSI



(a)



(b)

Figure 3.2: Transmission current under different RSSI values for: (a) WiFi and (b) cellular networks.

value and network throughput are collected by the context profiler, which incurs small energy overhead as we will show in Sec. 3.4.

We first derive the mapping between the RSSI and drawn current via experiments using an Android phone. More specifically, we use an Agilent 66321D power meter to measure and record the current of an HTC Sensation XE phone. We also implement an Android application to record the average signal strength throughout each experiment. Via placing mobile phone at different locations, we measure the current consumed by each data transfer of a 50 MB file over 3G and WiFi networks under different RSSI values. We repeat the experiment three times at each location and report the results in Fig. 3.2. This figure clearly shows that the drawn current is a decreasing function on the RSSI value, which make sense as better signal strength means that the mobile device can reduce the transmission power without increasing the bit-error-rate.

We fit the measurements to a polynomial function with the least square method. Table. 3.1 shows the Mean Square Error (MSE) of several polynomial functions. This table shows that the error of a cubic polynomial function is fairly low, and thus we use the cubic

Table 3.2: The Empirically-derived Cubic Model Parameters

Para.	γ_3	γ_2	γ_1	γ_0
Cellular	-1.35×10^{-5}	-2.9×10^{-3}	-0.21	-4.89
WiFi	-4.37×10^{-7}	-5.62×10^{-5}	-2.7×10^{-3}	0.19

polynomial function in our energy model:

$$P_{trans}(S) = \gamma_3 \times S^3 + \gamma_2 \times S^2 + \gamma_1 \times S + \gamma_0, \tag{3.4}$$

where γ_3, γ_2, γ_1, and γ_0 are the parameters of the cubic function, and S is the RSSI value in dBm. The model parameter derived on the HTC Sensation XE phone is given in Table 3.2.

With the RSSI and throughput collected by the context-aware profiler, we can write the communication energy consumption as:

$$E_{trans}(S, R, D) = P_{trans}(S) \times \frac{D}{R} \times V, \tag{3.5}$$

where R denotes the throughput and D denote the transferred data amount if the subject method is offloaded to the cloud. The energy model given in Eq. (3.5) is combined with the computation, display, and other power models defined in PowerTutor for a device-level energy model. This context-aware energy model is employed by the CADA algorithm to estimate energy consumption.

3.2.2 Application Modification without Source Code

How to offload the existing applications is also a significant issue. We want not only self-developed applications, but also existing applications can use offloading to reduce energy consumption. However, most of the time we do not have the source code of the applications. Without source code, we cannot easily determine what methods can be offloaded to cloud and modify the code to offload the methods. To help us easily analyze what methods can be offloaded and modify the applications without source code, we develop an APK analysis tool to achieve it. The tool analyzes APK file, which is the application installation file that used in Android OS, and modify it to use offloading library.

Fig. 3.3 shows the workflow of the tool. The first step is to decompile the APK file by apktool [2]. The decompiled files are Smali code, which is the register language of Dalvik Virtual Machine. Fig. 3.4 is an example of Smali code. It is not as readable as Java code but contains the information of super class, fields, and methods that can be used to analyze the application. We analyze the Smali files and filter out the methods that cannot be offloaded to cloud. Since local resources, such as camera and GPS, can only

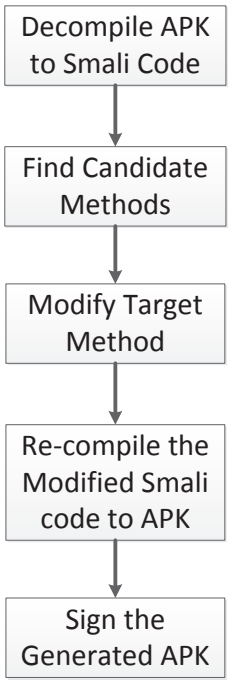


Figure 3.3: The workflow of the APK analysis tool.

accessed locally, we remove the methods and classes that access the local resources. Once the analysis done, we get a list of candidate methods that can be offloaded to cloud.

By assigning a target method, the tool automatically transforms the method to offload version. The tool creates a new Android project and an empty method, which has the same input, output, and name as the target method. Since we use ThinkAir to be our offloading library, we use ThinkAir code generator to make the created project use ThinkAir library. The Smali code of the target method then overwritten with the Smali code that decompiled from the created project. After modifying the smali code, it is recompiled to APK file through apktool. After signing the generated APK file, it can be installed in normal Android devices and offloading the target method to cloud.

3.3 Proposed Algorithm

While offloading computation to the cloud may save energy, it is not always true. For instance, if we offload a method to the cloud, when the wireless signal strength is low, the additional communication energy may be higher than the saved computation energy. Hence, a context-aware decision engine is critical to the system performance.

CloneCloud [9] and MAUI [11] solve integer linear programming problems in their decision engine, while ThinkAir [17] averages the historical execution costs on the cloud server and on the mobile device, and chooses the better one. Namboodiri and Ghose [23]

```
.class public Lchesspresso/game/Game;

.implements Ljava/io/Serializable;
.super Ljava/lang/Object;
.source "Game.java"

# interfaces
.implements Lchesspresso/position/PositionChangeListener;


# static fields
.field private static DEBUG:Z


# instance fields
.field private m_alwaysAddLine:Z

.field private m_changeListeners:Ljava/util/List;

.field private m_cur:I

.field private m_header:Lchesspresso/game/GameHeaderModel;

.field private m_ignoreNotifications:Z

.field private m_model:Lchesspresso/game/GameModel;

.field private m_moves:Lchesspresso/game/GameMoveModel;

.field private m_position:Lchesspresso/position/Position;


# direct methods
.method static constructor <clinit>()V
    .locals 1

    .prologue
    .line 55
    const/4 v0, 0x0

    sput-boolean v0, Lchesspresso/game/Game;->DEBUG:Z

    return-void
.end method
```

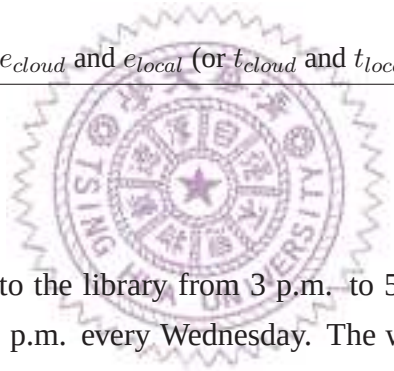
Figure 3.4: Example of Smali code.

also propose an algorithm to determine whether running an application in the cloud is more energy-efficient. Wolski et al. [29] implement several bandwidth measurement methodology and compares their performance in grid computing offloading. None of the works [9, 11, 17, 23, 29] make the offloading decisions based on rich contexts. In contrast, we design a Context-Aware Decision Algorithm (CADA) that takes four contexts into considerations, while making the offloading decisions.

In the CADA algorithm, we avoid the optimization problem solvers, used in, e.g., [9, 11], as solving complex integer linear programming problems itself may consume excessive energy. Instead, we use the time-of-day and location to determine whether offloading can save energy. We choose these two contexts because mobile users usually have a regular behavior [15]. This means that for a given time-of-day, a user is likely to visit the same place, have the same wireless network condition, and run the same applications. For

Algorithm 1 Context-Aware Decision Algorithm (CADA)

```
Query from database with T and L
if e_cloud is null then
    Query with L for the closest record
    if there is no record then
        Return offload to the cloud server
    end if
    Set t_cloud and e_cloud to the value of closest record
end if
if e_local is null then
    Query with L for the closest record
    if there is no record then
        Return execute on the mobile device
    end if
    Set t_local and e_local to the value of closest record
end if
Choose the smaller one of e_cloud and e_local (or t_cloud and t_local)
```



instance, for a user who go to the library from 3 p.m. to 5 p.m. every Tuesday and go to the park from 6 p.m. to 8 p.m. every Wednesday. The wireless network condition in library is usually good enough for offloading, but the wireless network condition in park is usually not. Thus, the CADA algorithm learns from the past execution records and offloads the workload if the time is between 3 p.m. and 5 p.m. on Wednesdays and the location is the library.

Before invoking a method which can be offloaded, the CADA algorithm uses the current location and time-of-day to query the past execution costs at the same location and time-of-day on the cloud server and on the mobile device. If there is no previous result in database, the CADA algorithm chooses the closest time-of-day at that location if it exists. Otherwise, the CADA algorithm tries to offload the method once and run the same method on the mobile device once to get some records for future, educational decisions. Algorithm 1 summarizes the CADA algorithm. T and L represent time-of-day and location, respectively. t and e are the execution time and energy consumption of the subject method. t_{local} is the execution time on the mobile device and t_{cloud} is the execution time on the cloud server. For energy consumption, we define e_{local} and e_{cloud} for that on the mobile device and the cloud server.

Table 3.3: Four Classes of Mobile Applications Used in the Experiments

Class	High Computation	Low Computation
Big State	HCBS	LCBS
Small State	HCSS	LCSS

3.4 Experiments

3.4.1 Offloading Simulation

We conduct real experiments to evaluate the proposed CADA algorithm and mobile cloud offloading system.

Setup

We have implemented our context-aware decision engine on Android 4.0, and integrated it with ThinkAir [17] for experiments. We deploy the client on multiple HTC phones and install the server on Android-X86 running on a commodity Intel i7 desktop. We ask 4 users, who are graduate students, to carry the smartphones and execute the offloading experiments whenever they get a chance. We only get enough experimental samples from four users, and thus exclude the two outliers in the rest of this section. We configure the profiler to cluster the GPS locations by rounding the latitude and longitude values to the fourth digits after the decimal point. The profiler collects the GPS locations once every 30 mins and other contexts once every 5 mins.

We consider four different classes of mobile applications, which demand for diverse computation and communication resources. The four kinds are: (i) Low Computation with Small State (LCSS), (ii) Low Computation with Big State (LCBS), (iii) High Computation with Small State (HCSS), and (iv) High Computation with Big State (HCBS). Table. 3.3 summarizes the classification. We believe that these four application classes represent most of the existing mobile applications. We implement a sample mobile application in each class, and upload the mobile applications to the HTC phones.

For LCSS, we implement a simple for-loop performing addition/multiplication operations. The amount of computations is light for modern smartphones and the state to be transferred is small. For HCSS, we implement nested for-loops with huge numbers of iterations performing addition/multiplication operations. The amount of computations is large and takes the HTC phones about 30 secs to finish. For LCBS, the application reads an image and performs a simple color space conversion from the RGB to YUV space. This application transfers a high-resolution image to the cloud and the computation de-

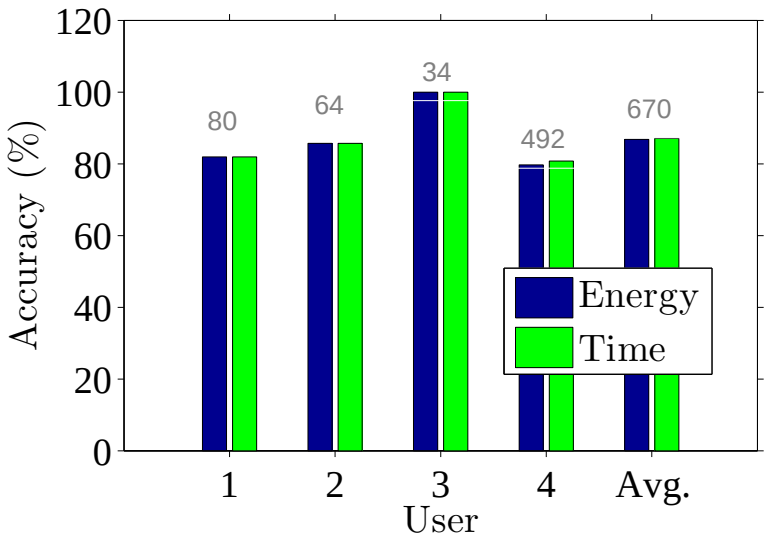


Figure 3.5: Accuracy among users, each user is annotated by number of offloading experiments.

mand is low. Last, for HCBS, we perform the same as LCBS but added a face detection algorithm, which leads to large computation demand.

The subjects take the HTC phones with them for 3 days, and run random mobile applications whenever they get a chance. We consider three performance metrics: (i) prediction accuracy, (ii) performance gain in the energy consumption and response time, and (iii) energy and computation overhead.

Results

Importance of context-aware offload decision. Different applications have diverse features and thus their behaviors are not identical. We build a simple experiment to demonstrate the negative impacts of making offloading decisions without considering user contexts. In particular, we execute the HCBS application with WiFi, 3G, and local respectively, and give the results in Table 3.4. In this table, Idle is the energy consumption without our offload-able application. The result we report is the additional energy consumption compare with idle energy. This table shows that offloading the application over 3G leads to higher overall energy consumption on the mobile device, compared to running the same application on the mobile device. This reveals that cloud offloading could result in worse performance when the network condition is bad. In other words, the network condition imposes a direct implication on offloading performance. Given that offloading this application with 3G at this location is worse than performing that on the mobile device, our CADA algorithm will make a smart decision to run that application on the

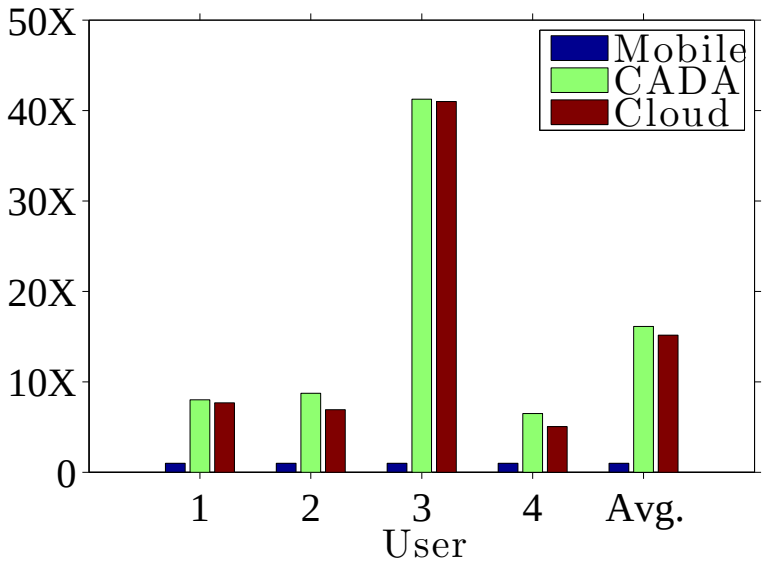


Figure 3.6: Performance improvement on response time.

Table 3.4: Energy Consumption when Offloading over Different Wireless Networks

Network Type	**WiFi**	**3G**	**Local**	**Idle**
Current (mA)	9.2619	285.3662	24.0448	398.74
Execution Time	224.99	3559.85	299.34	-

mobile device.

Accuracy of the CADA algorithm. We present the decision accuracy of the CADA algorithm in Fig. 3.5. The accuracy is defined as the percentage of making the right decisions for lower energy consumption or shorter response time; whereas the right decisions are determined by comparing against the ground truth collected by the context-aware profiler. Since CADA enforces each method to execute on the mobile device and on the cloud server once to learn which one is the better, we refer to these two executions as the training round and exclude them when we compute the accuracy. Fig. 3.5 depicts that among the four users, the accuracy is between 79% to 100%. The user who achieves 100% accuracy tends to stay in very few locations, and thus the CADA algorithm can make better prediction using the larger number of samples. Another important observation is that the CADA algorithm can optimize toward lower energy consumption and shorter response time, and achieve roughly the same accuracy. This demonstrates the flexibility of our CADA algorithm and mobile cloud offloading system. Fig. 3.8 and 3.9 show the energy consumption and execution time of each scenario. The figures present that time and energy are proportional and most of the scenarios have big cost difference, which is at least

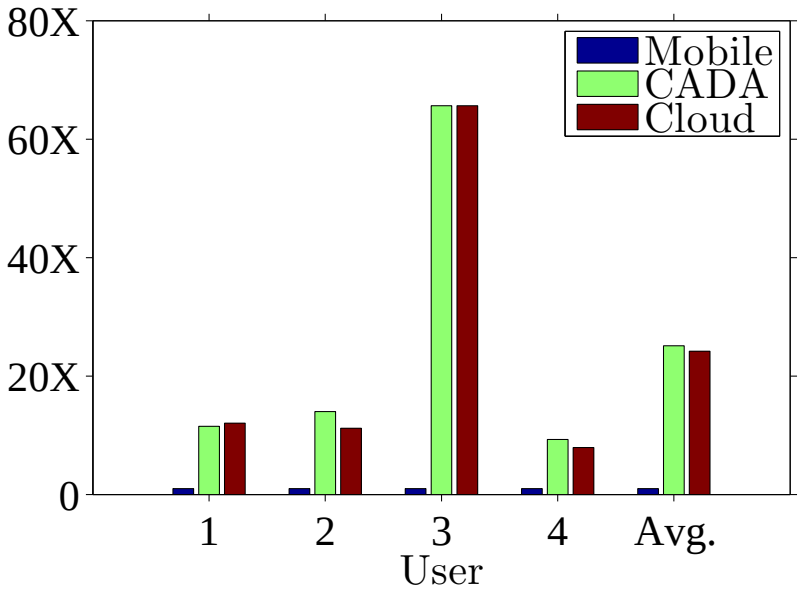


Figure 3.7: Performance improvement on energy consumption.

40%, between executing on local and cloud. Since the pattern of time and energy are similar, the accuracy of time and energy are very similar as well.

Performance gain achieved by the CADA algorithm. Figs. 3.6 and 3.7 plot the performance improvements on response time and energy consumption achieved by the CADA algorithm, respectively. The results are normalized to the results achieved by running the mobile applications on the mobile devices. These figures show that the proposed CADA algorithm outperforms the baseline approach in most cases. The only exception is the energy consumption of user 1 in Fig. 3.7, which consumes slightly more energy compared to always running the mobile applications on the cloud server: a mere 4% gap is observed. In summary, the CADA algorithm improves the mobile cloud offloading performance in response time or energy consumption for 87.5% users.

Overhead. CADA consumes very few CPU cycles, and terminates in less than 1 ms for all users. The number of entries in the context-aware database is no more than $\mathbf{M} \times \mathbf{L} \times 336$, where $\mathbf{M}$ is the number of methods to be offload and $\mathbf{L}$ is the number of locations where the users execute those methods. 336 is derived by 48 time slots each day and 7 days a week. This number is fairly manageable on modern mobile devices.

We design a simple experiment to measure the overhead of our profiler. We use an Agilent 66321D power meter to measure the average current of an HTC sensation XE phone with and without our profiler for 1 hour. In fact, our profiler collects many more contexts although we only need network RSSI, throughput, and GPS location in our experiments. To be conservative, we enable the profiler to collect all of the contexts. Table. 3.5 gives the measurement results. This table shows that the average gap is 2.94 mW, only 6% more

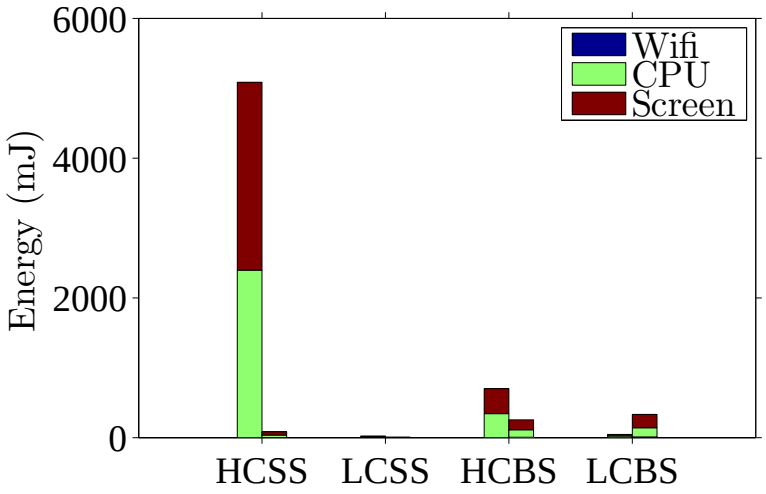


Figure 3.8: Energy consumption of CPU, display, and Wifi of each scenario.

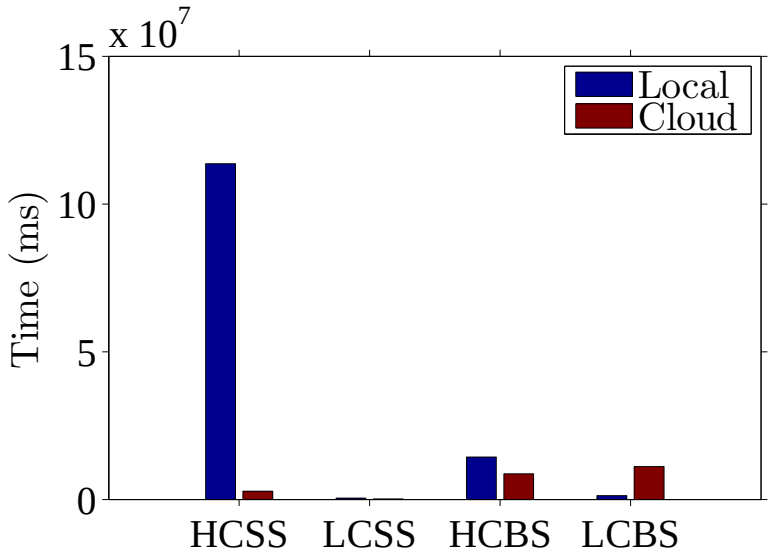


Figure 3.9: Execution time of each scenario.

Table 3.5: Profiler Energy Overhead

Power (mW)	Average	Min	Max
Without profiler	48.9	47.1	50.5
With profiler	51.84	46.9	56.3

than the current with profiler.

3.4.2 Offloading Real Third-party Applications

In this experiment, we aim to show that offloading can be used in existing third-party applications and improves performance or reduces energy consumption of mobile devices. Since the third-party applications usually don't have source code but only APK files, which can be used to install the application on mobile devices, we use our APK analysis tool to analyze the applications and modify the APK files.

We use HTC one X to be our mobile device and download four applications from Internet. The offloading server is a virtual machine with android-x86 running on a desktop PC. By utilizing the APK analysis tool, we analyze the applications to find the methods that can be offloaded to cloud. After the methods are found, we carefully choose target methods from the candidate methods and use the tool to modify the target methods. We install the modified APK files on the mobile device and trigger the modified methods ten times.

The execution time of local and offloading are shown in Table 3.6. Since we cannot find any candidate method in the third application, we use X to indicate that it does not have any candidate method. We observe that the first application (Pocket Chess) reduces 34% execution time with offloading. This observation shows that offloading can be used in existing applications and improves the application performance. However, as mentioned before, offloading is not always true. There are two applications that do not have improvement with offloading. The execution time of offloading are 10 times longer than local. Therefore, offloading decision is necessary and should carefully determine whether to offload.

Table 3.6: Energy Consumption when Offloading over Different Wireless Networks

Application Name	**Local (ms)**	**Offloading (ms)**
Packet Chess	2641.5	1742.5
Chess Road	7.9	89.4
Alien Invasion	X	X
Pin Ball	12.7	180.3

Chapter 4

Offloading Event Analysis Algorithms: Using IsCrowded and IsNoisy Events As Case Studies

In this chapter, we discuss how we integrate offloading into a crowdsensing system and show the offloading benefit with case studies.

4.1 Implementation

4.1.1 System Overview

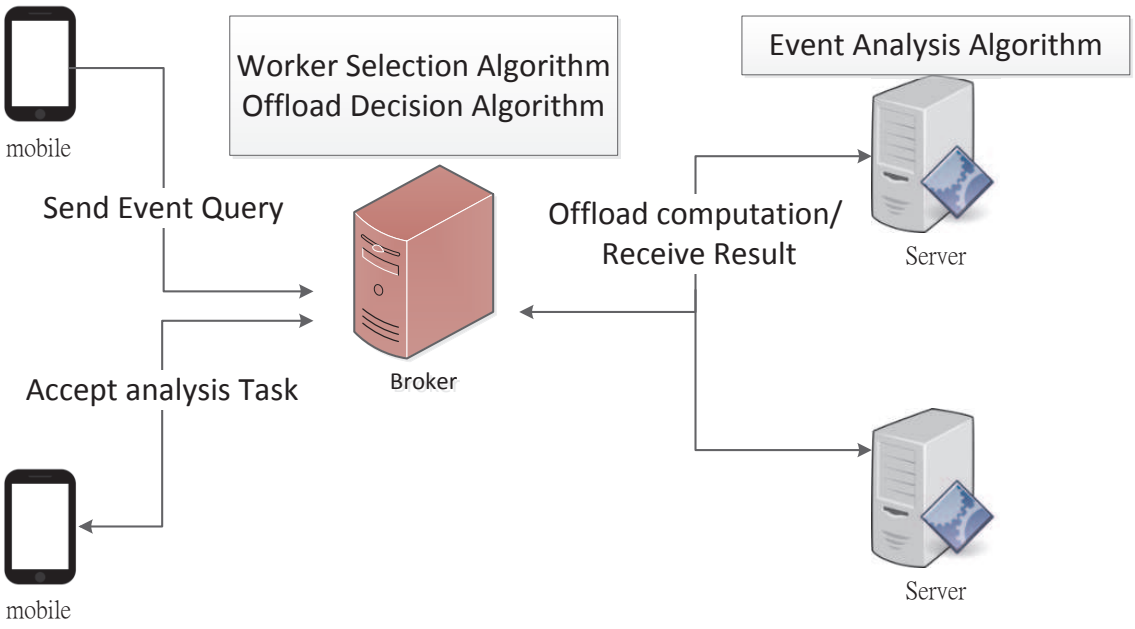


Figure 4.1: Overview of the crowdsensing system.

Fig. 4.1 presents the overview of the crowdsensing system. Mobile users submit queries to the broker or accept query tasks from broker. A query insists of (i) event, (ii) required location, (iii) time period, and (iv) location of the mobile device. Event is the desired event that the mobile user wants to know, for example, whether the required location is crowded or not. The events are detected by event analysis algorithms, which analyze the collected sensory data to determine whether the event happen or not. Since event analysis algorithms may lead to heavy computation and energy consumption, we use offloading to reduce the overhead of mobile devices. For the mobile users who accept the queries, they move to the required locations and collect the sensory data within the time period. Broker is a logically controller managing the connection of the mobile users. It maintains all the queries that are not satisfied and recommend the queries to mobile users to perform. The decision algorithm is implemented in broker since broker is logically controller and has the context of the servers and mobile devices. When offloading happens, broker becomes the bridge of the mobile devices and servers, which are responsible for executing the offloaded event analysis algorithms.

Figs. 4.2–4.4 show the screenshots of the client side user interface. Through binding the required location in the red rectangle as shown in Fig. 4.2, users can easily choose the required location. Once the event, required location, and time are selected, users can submit their queries by click on the submit button. The queries that are not satisfied are shown on the map (see Fig. 4.3). Users click on the markers to see the details of the queries and decide whether to accept the task of the clicked query. By clicking the sense button in Fig. 4.4, mobile devices start to collect the sensory data required for analyzing the events. Once the sensory data are collected, broker gives instructions to instruct whether execute the event analysis algorithms on mobile devices or offload to cloud.

4.1.2 Prototype Architecture

We present the architecture of the prototype in Fig. 4.5.

Android Client. The front-end user interface provides easy and straightforward interface for users choosing the event, required location, and time as shown in Figs. 4.2–4.4. GPS listener monitors the location of mobile devices and provide the location information when submit/answer the queries. Broker connector responsible for all the communications between broker and client. Once offloading be triggered, the data are sent through broker connector and wait for the results. Event analysis algorithms are implemented in client side. Through the offloading library, the event analysis algorithms can be executed both on mobile devices and cloud servers.

Broker. Since there are many clients connect to broker, there is a client manager to manage the clients. The decision algorithm is implemented in client manager to de-

Figure 4.2: Screenshot of query tab.

Figure 4.3: Screenshot of task tab.

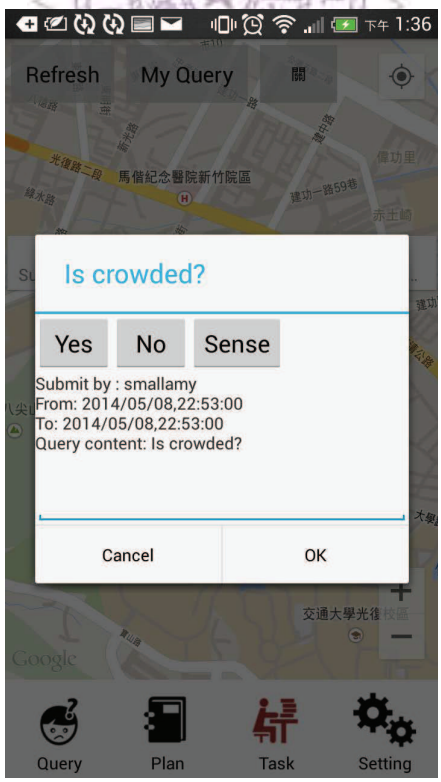


Figure 4.4: Screenshot of answer dialog.

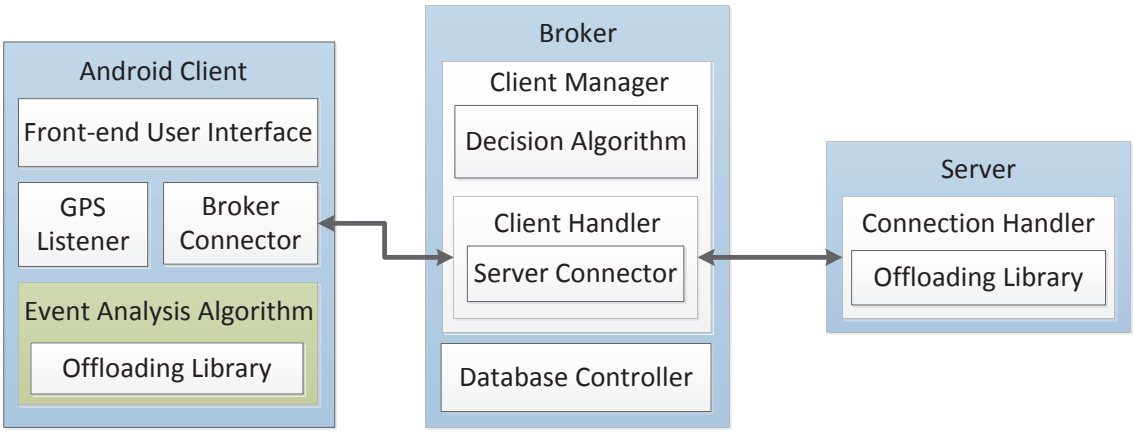


Figure 4.5: Prototype architecture.

termine whether clients offload their event analysis algorithms to cloud. Each client is handled by a client handler, and it communicates with the broker connector that is on client side. Client handler handles all the requests from clients, except for offloading. Offloading requests are given to server connector to proceed the offloading process. We use MySQL [3] to be our database and broker communicates with the database through the database controller. The database maintains all the queries, answers, and user contexts.

Server. Servers are responsible for executing the event analysis algorithms. It has connection handler to handle multiple offloading requests, and executes the offloading request through the offloading library. The answers are sent back to clients through broker once the event analysis algorithms are finished.

4.2 Case Study

In this section, we use two event analysis algorithms to be our case studies. The events are (i) IsCrowded and (ii) IsNoisy.

IsCrowded. IsCrowded is the event that users want to know whether the required location is crowded with people. In order to understand whether is crowded or not, we use image processing with human detection to achieve it. The most popular techniques for human detection are face detection and histograms of oriented gradients (HOG) [12]. Face detection computes the human face features appeared in the images to determine how many people are in the images. However, face detection requires that all the people in the images should face to the camera otherwise cannot be detected. Therefore, we use another technique called histograms of oriented gradients (HOG) [12] instead.

Fig. 4.6 presents the work-flow of HOG human detection. The input images are scaled to proper size to be processed by the HOG algorithm, and converted to gray-scale images

and perform color space normalization to reduce the impact of shadow and noise. The detection window is shifting on the images and computes the gradients of the covered area at each run. The result of gradients are given to a support vector machine (SVM) classifier to detect whether human appeared. In our implementation, we set the window size to 8×8, and use the SVM and parameters trained by openCV [4] to implement human detection algorithm.

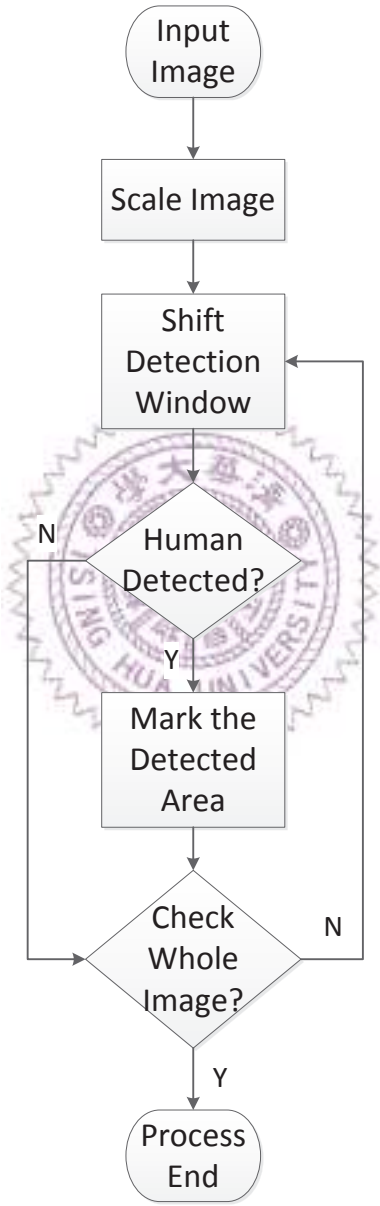


Figure 4.6: HOG human detection workflow.

To show the detected humans in the images, we use rectangles to mark the detected humans. Fig. 4.7 is an example of using HOG to detect human in an image, and the detected humans are marked with rectangles. We can see that HOG detects humans in the image and its accuracy is high. However, the complexity of computing the gradients is a

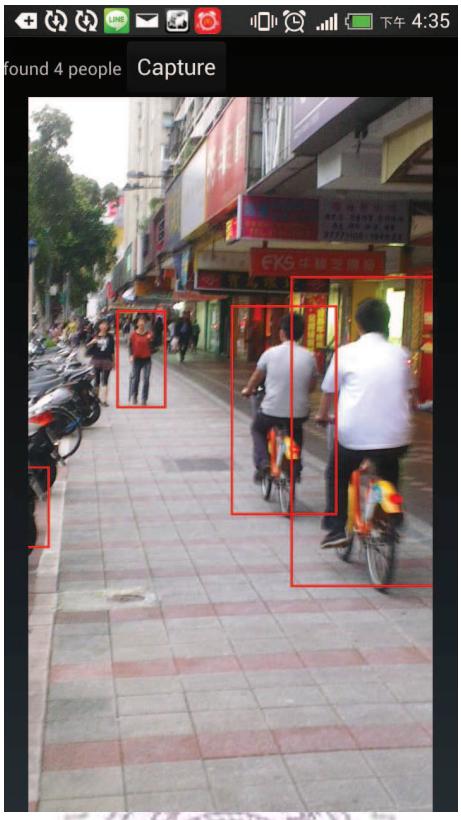


Figure 4.7: Human detection using HOG.

heavy overhead even using native code to implement it. This can lead to long execution time and high energy consumption on mobile devices. The execution time and energy are shown in next section.

IsNoisy. For IsNoisy event, we are determining whether the required location is noisy or not. We record the noise through microphone, and process the recording file to get the average decibel (db) to determine whether the environment is noisy or not.

4.3 Evaluations

4.3.1 Setup

To understand the performance improvement of the prototype system with offloading, we conduct experiments to observe the results. We use HTC one X to be our mobile device. The broker and server are running on VMs, which are hosting on a desktop PC. In order to deal with android classes, the VMs are Android-x86 machines. We setup a PC located between the broker and mobile client with dummynet. The network condition can be easily controlled by the dummynet, and we vary network delay, packet loss rate, and bandwidth to observe how network condition affects the offloading results. Network delay

Table 4.1: Ideal Case Performance of IsCrowded Event

IsCrowded	Min	Max	Avg.
Local execution Time (ms)	31322	33389	31957
Offload execution Time (ms)	15536	19924	18062
Local energy (mJ)	10474	10887	10618
Offload energy (mJ)	2396	2992	2759.2

is varied in $[10, 50, 100, 200, 400]$ ms, packet loss rate is varied in $[0, 2, 4, 8, 10]\%$, and bandwidth is varied in $[256, 512, 1024, 2048, 4096]$ kbit/s. If not specified, we let network dealy = 10 ms, packet loss rate = 0%, and bandwidth = 4096 kbit/s. For each event, we execute the event analysis algorithm 5 times at cloud. Since network condition does not affect the performance of local execution, we execute the event analysis algorithms 5 times at local and use it as baseline.

4.3.2 Results

Ideal case. We present the ideal case of offloading the event analysis algorithms to understand whether the event analysis algorithms have performance gain through offloading. We set network delay = 10 ms, packet loss rate = 0%, and bandwidth is unlimited. The ideal case results are shown in Table 4.1 and 4.2. We observe that offloading improves the performance of the crowdsensing prototype system due to leverage the powerful cloud server to execute the heavy computation of HOG human detection. Offloading reduces the processing time and energy consumption of HOG to 56.5% and 25.9% compared to processing on mobile device. However, offloading the event analysis algorithm of IsNoisy to cloud lead to worse processing time and energy. It takse 2.35 times processing time and 1.84 times energy consumption compare to local execution. The ideal case results present that offloading improves the crowdsensing system for IsCrowded event, but IsNoisy event does not have performance gain through offloading. Therefore, we only show the performance impact of IsCrowded event while varying network condition and ignore IsNoisy event.

Impact of network delay. Fig. 4.8 shows that longer network delay leads to longer execution time. Longer network delay leads to longer transmission time, therefore, it takes longer transmission time for mobile devices to transfer the data. We observe that it has performance gain when the network delay is less than 50 ms. However, it takes 2.57 times execution time when the network delay is 400 ms. The energy consumption is shown in Fig. 4.9. Longer network delay leads to larger energy consumption. When the

Table 4.2: Ideal Case Performance of IsNoisy Event

IsNoisy	Min	Max	Avg.
Local execution Time (ms)	123	196	162.2
Offload execution Time (ms)	332	474	381.2
Local energy (mJ)	23	46	37
Offload energy (mJ)	54	75	68.3

network delay is less than 200 ms, offloading is better. Once the network delay is 400 ms, offloading results in 1.28 times energy consumption compare to local execution.

Impact of packet loss rate. The execution time is reduced when packet loss rate is less than 4%, as shown in Fig. 4.10. Higher packet loss rate leads to more retransmission of the packets, therefore, the execution time is longer. The execution time is 2.52 times higher than execute on mobile when packet loss rate is 10%. Fig. 4.11 shows the energy consumption while varying the packet loss rate. It benefits from offloading when packet loss rate is less than 10%. If the packet loss rate is not smaller than 10%, we do not have performance gain through offloading.

Impact of bandwidth. Bandwidth affects the transmission time. It takes longer transmission time to send data when bandwidth is low. Fig. 4.12 presents the execution time under different bandwidth. We observe that IsCrowded event requires bandwidth higher than 1024 kbit/s for shorter execution time. If bandwidth is only 256 kbit/s, offloading takes 3.13 times execution time compare to local execution. Fig. 4.13 presents the energy consumption under different bandwidth. It shows that 512 kbit/s is sufficient for energy gain.

The results present that offloading improves the execution time and energy consumption if we carefully determine whether to offload. Different network conditions lead to different performance gains. In order to improve the performance of crowdsensing systems through offloading, offloading decision is necessary.

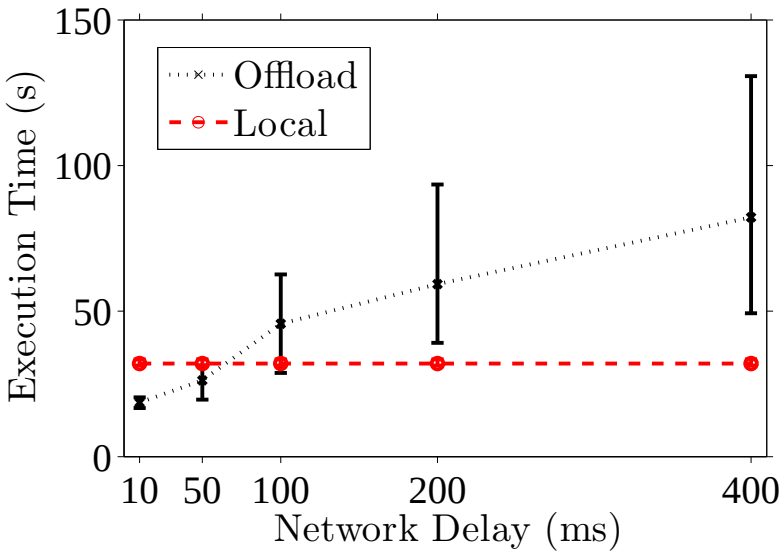


Figure 4.8: Execution time while varying network delay.

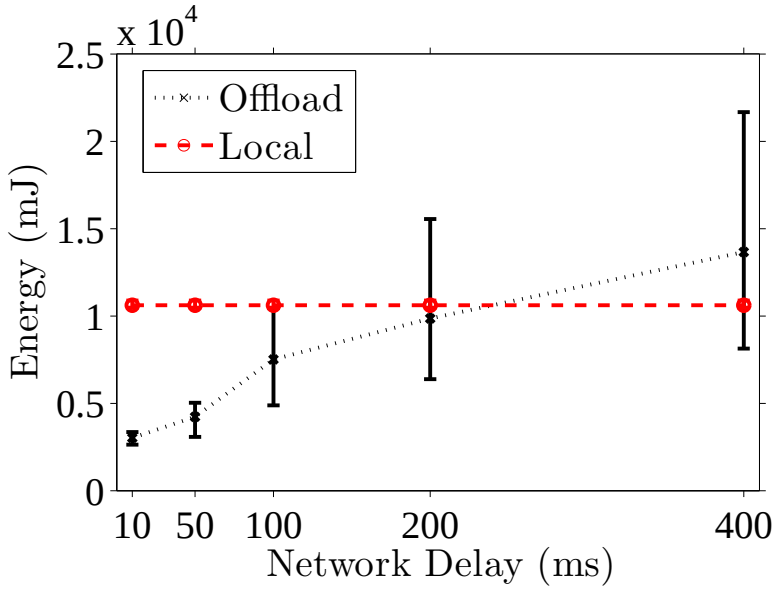


Figure 4.9: Energy consumption while varying network delay.

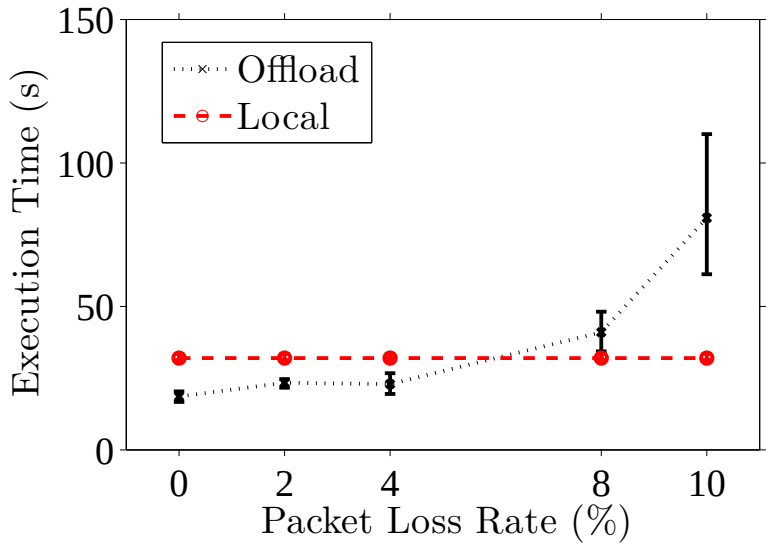


Figure 4.10: Execution time while varying packet loss rate.

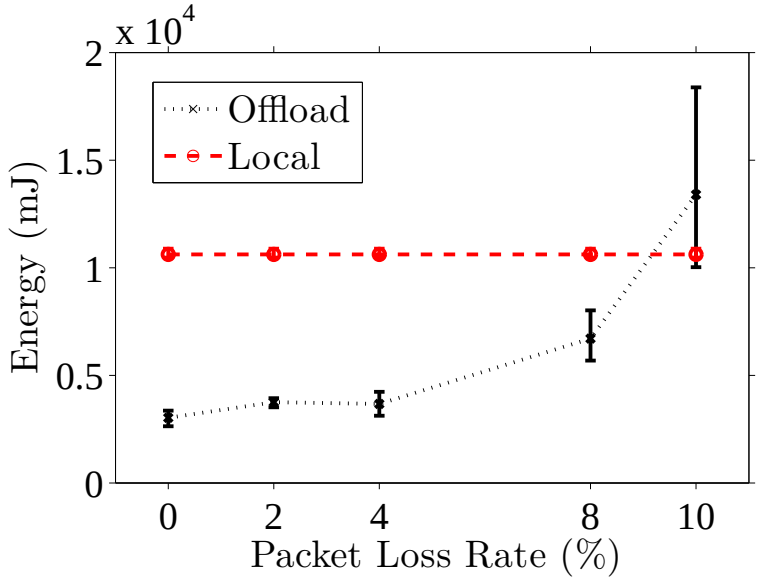


Figure 4.11: Energy consumption while varying packet loss rate.

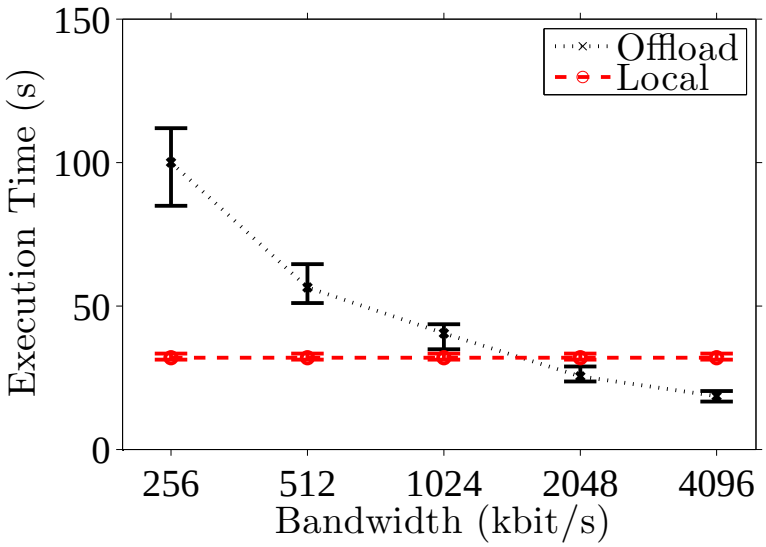


Figure 4.12: Execution time while varying bandwidth.

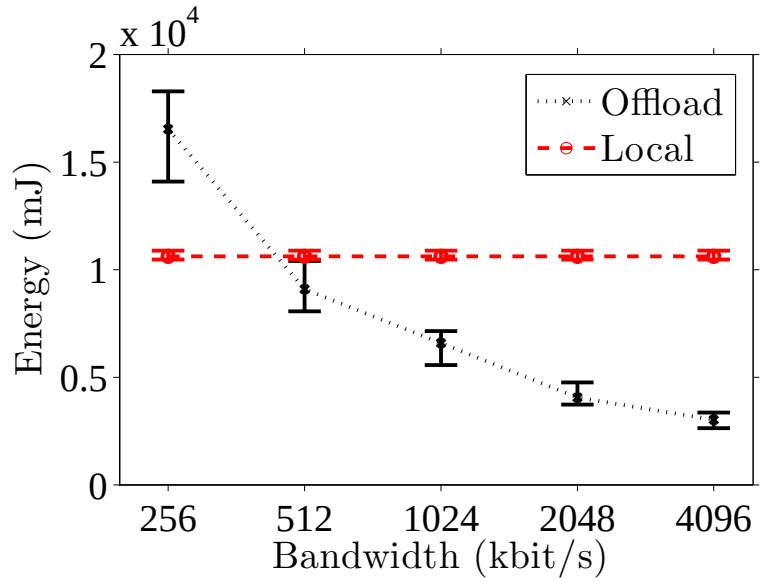


Figure 4.13: Energy consumption while varying bandwidth.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We proposed a context-aware decision algorithm, called CADA, which uses the location and time-of-day to make the mobile offloading decisions of individual methods. We developed a tool to analyze and modify the existing applications without source code. In order to study the performance gain in real crowdsensing system, we implemented event analysis algorithms and integrated offloading library into the crowdsensing prototype system.

The CADA algorithm is flexible and supports different optimization criteria, including minimizing response time and minimizing energy consumption. The CADA algorithm can work with the mobile cloud offloading systems proposed in the literature. We have implemented the CADA algorithm, integrated it with ThinkAir, and conducted experiments using several HTC phones. Our experimental results show that the CADA algorithm achieves more than 80% prediction accuracy, and leads to better performance in terms of response time and energy consumption. The complexity of the CADA algorithm is no more than 0.011 ms. We also proposed a context-aware energy model for mobile devices to accurately measure the energy consumption. The result of modifying existing applications shows that offloading can be used in existing applications to reduce the energy consumption or execution time for mobile devices. Our experiments also show that offloading improves the performance of crowdsensing system through offloading the overhead of event analysis algorithms to cloud servers.

5.2 Future Work

There are several directions that we plan to address in the future.

- We plan to introduce additional user contexts that may further increase the decision

accuracy of our CADA algorithm. For example, we may introduce the usage history of applications of users to further understand the user behavior.

- A hierarchical decision engine to reduce the overhead of always getting decisions from broker. That is, mobile devices execute light weight local decision or cache the decision. If the accuracy of local decision is unsatisfied, mobile devices ask broker for more accurate decision.

- Our algorithm can be used for offloading computations to multiple servers in crowd-sensing systems. Optimally determining where to offload each method is among our future tasks.

- We plan to collect a more realistic dataset for evaluating our algorithms. The dataset we have is not collected from real applications. We will develop real applications and our users can use it in their daily life. Therefore, the collected dataset is more realistic and represents user's daily usage behavior.

- To improve the performance of our CADA algorithm, we will collect more contexts and fine-grained data log in the future.

- Current mobile cloud offloading systems only support GUI-less workloads. We are looking into the possibility of offloading general mobile applications with local resources (e.g. camera and GPS). This will relieve the burden of using offloading for developers.

Bibliography

[1] 2012 U.S. wireless smartphone and traditional mobile phone satisfaction studies. http://www.jdpower.com/content/press-release/py6kvam/.

[2] apktool. https://code.google.com/p/android-apktool/.

[3] Mysql. http://www.mysql.com/.

[4] Opencv. http://opencv.org/.

[5] Powertutor. http://ziyang.eecs.umich.edu/projects/powertutor/.

[6] V. Agarwal, N. Banerjee, and D. Chakraborty. Usense - a smartphone middleware for community sensing. In *Proc. of IEEE International Conference on Mobile Data Management (MDM'13)*, pages 52–65, Milan, Italy, June 2013.

[7] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC'09)*, pages 280–293, Chicago, IL, USA, November 2009.

[8] Y. Chon, N. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proc. of ACM Conference on Ubiquitous Computing (UbiComp'12)*, pages 481–490, Pittsburgh, Pennsylvania, September 2012.

[9] B. Chun, M. Naik, S. Ihm, A. Patti, and P. Maniatis. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of European Conference on Computer Systems (EuroSys'11)*, pages 181–194, Salzburg, Austrial, April 2011.

[10] V. Coric and M. Gruteser. Crowdsensing maps of on-street parking spaces. In *Proc. of IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'13)*, pages 115–122, Cambridge, MA, May 2013.

[11] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 49–62, San Francisco, CA, USA, June 2010.

[12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 886–893, San Diego, CA, USA, June 2005.

[13] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*, pages 335–348, Washington, DC, USA, June 2011.

[14] R. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: Current state and future challenges. *IEEE Communication Magazine*, 49(11):32–39, November 2011.

[15] M. Gonzalez, C. Hidalgo, and A. Barabasi. Understanding individual human mobility patterns. *Nature*, 453:779–782, June 2008.

[16] D. Hasenfratz, O. Saukh, S. Sturzenegger, and L. Thiele. Participatory air pollution monitoring using smartphones. In *Proc. of International Workshop on Mobile Sensing*, Beijing, China, April 2012.

[17] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of IEEE INFOCOM*, pages 945–953, Orlando, FL, USA, March 2012.

[18] Y. Kwon and E. Tilevich. Power-efficient and fault-tolerant distributed mobile execution. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'12)*, pages 586–595, Macau, China, June 2012.

[19] K. Lan, C. Chou, and H. Wang. An incentive-based framework for vehicle-based mobile sensing. *Procedia Computer Science*, 10:1152–1157, 2012.

[20] C. Liao, T. Hou, T. Lin, Y. Cheng, A. Erbad, C. Hsu, and N. Venkatasubramania. Sais: Smartphone augmented infrastructure sensing for public safety and sustainability in smart cities. In *Proc. of International Workshop on Emerging Multimedia Applications and Services for Smart Cities(EMASC'14)*, pages 3–8, Orlando, Florida, USA, November 2014.

[21] T. Lin, T. Lin, C. Hsu, and C. King. Context-aware decision engine for mobile cloud offloading. In *Proc. of IEEE Wireless Communications and Networking Conference Workshops (WCNCW'13)*, pages 111–116, Shanghai, China, April 2013.

[22] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proc. of Annual International Conference on Mobile Computing and Networking (Mobicom'12)*, pages 317–328, Istanbul, Turkey, August 2012.

[23] V. Namboodiri and T. Ghose. To cloud or not to cloud: A mobile device perspective on energy consumption of applications. In *Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'12)*, pages 1–9, San Francisco, CA, USA, June 2012.

[24] G. Perrucci, F. Fitzek, and J. Widmer. Survey on energy consumption entities on the smartphone platform. In *Proc. of IEEE Vehicular Technology Conference (VTC Spring'11)*, pages 1–6, Budapest, Hungary, May 2011.

[25] M. Ra, B. Liu, T. Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*, pages 337–350, Lake District, UK, June 2012.

[26] A. Rahmati and L. Zhong. Context-based network estimation for energy-efficient ubiquitous wireless connectivity. *IEEE Transactions on Mobile Computing*, 10(1):54–66, January 2011.

[27] W. Sherchan, P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke, and A. Sinha. Using on-the-move mining for mobile crowdsensing. In *Proc. of IEEE International Conference on Mobile Data Management (MDM'12)*, pages 115–124, Bengaluru, Karnataka, India, July 2013.

[28] M. Talasila, R. Curtmola, and C. Borcea. Improving location reliability in crowd sensed data with minimal efforts. In *Proc. of Joint IFIP Wireless and Mobile Networking Conference (WMNC'13)*, Dubai, United Arab Emirates, April 2013.

[29] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. Using bandwidth data to make computation offloading decisions. In *Proc. of International Parallel & Distributed Processing Symposium (IPDPS'08)*, pages 1–8, Miami, FL, April 2008.

[30] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of IEEE/ACM/IFIP International Conference on*

Hardware/Software Codesign and System Synthesis (CODES/ISSS'10), pages 105–114, Scottsdale, AZ, USA, October 2010.

[31] P. Zhou, Y. Zheng, and M. Li. How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'12)*, pages 379–392, Lake District, UK, June 2012.

Symbol Table

Symbol	Description
P_{total}	Total energy consumption
P_{cpu}	The energy consumption of CPU
P_{comm}	The energy consumption of wireless communication
$P_{display}$	The energy consumption of screen
P_{other}	The energy consumption of other components
$P_{WiFi/Cell}$	The energy consumption of WiFi/Cellular
P_{idle}	The energy consumption while idling
P_{trans}	The energy consumption of transmitting data
β_{idle}	The fraction of time while idling
β_{trans}	The fraction of time while transmitting data
γ	The parameter of energy model
S	Signal strength
R	Throughput
D	Data size
V	Voltage
T	Time-of-day
L	Location
t_{local}	Execution time on mobile device
t_{cloud}	Execution time on cloud
e_{local}	Energy consumption on mobile device
e_{cloud}	Energy consumption on cloud
M	The number of methods to be offload
L	The number of locations