# TOWARDS SUSTAINABLE MOBILE COMPUTING FROM A LIFE CYCLE PERSPECTIVE

A Dissertation by

Toolika Ghose

Master of Engineering, University of Texas at Tyler, Texas, 2007

Bachelor of Technology, Uttar Pradesh Technical University, India, 2004

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

December 2013

TOWARDS SUSTAINABLE MOBILE COMPUTING FROM A LIFE CYCLE PERSPECTIVE

The following faculty members have examined the final copy of this dissertation for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Doctor of Philosophy with a major in Electrical Engineering.

_____
Ravi Pendse, Committee Chair

_____
Vinod Namboodiri, Committee Member

_____
M. Edwin Sawan, Committee Member

_____
John Watkins, Committee Member

_____
Linda Kliment, Committee Member

Accepted for the College of Engineering

_____
Vish Prasad, Interim Dean

Accepted for the Graduate School

_____
Abu S. M. Masud, Interim Dean

# DEDICATION

To my loving parents and my husband

ACKNOWLEDGEMENTS

This dissertation work represents a milestone in my academic career. I could not have reached my goals without the guidance, support, inspiration and advice from several people at Wichita State University (WSU). I take this opportunity to express my deepest appreciation towards everyone who aided me in completing this work.

Firstly, I like to sincerely thank my advisor Dr. Ravi Pendse and co-advisor Dr. Vinod Namboodiri for their support, guidance and mentorship during my program of study at Wichita State University. I am grateful to both of them for giving me the valuable opportunity to work independently on a novel and prospective research topic. I would always cherish and remember the valuable leadership and experiences offered to me during countless encounters with them. I would also like to thank my committee members Dr. M. Edwin Sawan, Dr. John Watkins, Dr. Linda Kliment for their guidance, feedback and valuable time.

I am also thankful to Dr. Ravi Pendse for giving me the invaluable opportunity to work at the Advanced Networking Research Institute (ANRI) at WSU while pursuing my graduate studies. This opportunity not only allowed me to efficiently utilize my management and leadership skills in a real-time environment but also facilitated to equip myself with a high level of technicality as a Network Engineer. My life as a student at WSU would not have been so much fun without the network of wonderful, supportive and loving people at ANRI. I would like to convey my special thanks to my manager Amarnath Jasti and all my colleges at ANRI.

Wichita State University also introduced me to many inspiring and generous people such as Dr. M. Edwin Sawan who established the Maha Maggie Sawan Fellowship for International Students. I have had the privilege of being the first ever winner of the Maha Maggie Sawan

Fellowship. I would like to express my deepest gratitude to Dr. Sawan for his generosity. I like to thank the WSU Graduate School and the Department of Electrical Engineering and Computer Science for their administrative and financial support during my program of study.

This dissertation would not have come to a successful finishing point without the help of my WSU friends Prithvi Manduva, Vijay Sankar Venkitachalam, Vijay Ragothaman, Vishnu Dev Cherusola and Babak Karimi who were kind enough to support me with many of their resources. I would also like to thank Milindkumar Tandel, Shubhangi Satras, and Jyothsna Nissankarao for their valuable time and helpfulness in developing the test applications for my experiments.

I like to extend my heartfelt gratitude to all the members of my family for their immense support during my journey at WSU. The unconditional love, encouragement and inspiration from my family were a motivating force behind my successful completion of this work. I owe so much gratitude towards my dear dad (Late.) Dr. Animesh.Ghose and mother Dr. Ranjana Ghose. *Thank you, Maa, for your prayers, love and blessings*. A special thanks both to my sister Dr. Tandra Ghose and brother-in-law Dr. Rupak Majumdar for all the love, inspiration and guidance. I am also lucky to have such a loving brother Nirnimesh Ghose and nephews Ritwik and Rajit who always cheer me with their affection and best wishes at all times. I like to specially thank my husband Nikhil for his remarkable patience and countless helpful suggestions towards my research and dissertation work. The incredible support and motivation from him have been the greatest assets for me to have remained focused on my goal through good and bad times. *It would not be possible to describe through words about how grateful I am to have you as my best-half.*

# ABSTRACT

The exponentially increasing demand for smartphones, tablets, and laptops has made the mobile communication sector one of the fastest-growing industries. Although features and functionality of these devices have improved with time, they still lack in competencies such as processing capability and memory storage, compared to personal desktop computers. Advancements in storage capabilities and infrastructure in cloud-based technologies could be beneficially used to overcome the limitations of mobile devices.

This dissertation analyzes the performance and energy consumption of mobile devices used under a cloud-based computing paradigm. This kind of client-server setup enables flexibility in the execution of applications either on the client end or the cloud server end based on the availability of resources at each end. This dissertation analyzes the theoretical limits and proposes optimal methods for energy-efficient application management in mobile devices.

Furthermore, in this dissertation, the impact on user experience owing to cloud-based applications is considered and quantified by proposing a user-experience metric, namely "AppScore," a comparison metric that evaluates the relative functionality and performance of applications. Finally, the environmental challenge posed by these mobile devices due to shorter life span is also considered. A comprehensive life-cycle energy (LCE) consumption model for mobile devices is proposed and validated by implementing it on a cloud-based, thin-client (simplified device with fewer hardware components) scenario.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF ABBREVIATIONS/NOMENCLATURE

| | |
|---|---|
| 3G | Third Generation |
| ADT | Android Development Tool |
| $CO_2$ | Carbon Dioxide |
| COTS | Common Off-the-Shelf |
| DNS | Domain Name Server |
| GB | Gigabyte |
| GHz | Gigahertz |
| GPS | Global Positioning System |
| GSM | Global System for Mobile |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| kB | Kilobyte |
| Kg | Kilogram |
| Km | Kilometer |
| LCD | Liquid Crystal Display |
| LCE | Life-Cycle Energy |
| MB | Megabyte |
| ms | Millisecond |
| mW | Milliwatt |
| mWatt-Sec | Milliwatt-Second |
| MWh | Megawatt Hours |
| NIC | Network Interface Card |

| | |
|---|---|
| OLED | Organic Light Emitting Diode |
| PSM | Power Save Mode |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| SD | Secure Digital |
| SDK | Software Development Kit |
| TCP | Transport Control Protocol |
| UDP | User Datagram Protocol |
| UI | User Interaction |
| UID | Unique Identification |
| URI | Uniform Resource Identifier |
| XML | Extensible Markup Language |

# CHAPTER 1

# INTRODUCTION

The progression in mobile device technology has introduced a great deal of mobility and global connectivity. The first phone call over Global System for Mobile Communications (GSM) occurred in 1991, and currently 77% of the global population uses cell phones. About 174 million smartphones were sold globally in 2009 [1, 2, 3]. Smartphones and other such smart mobile devices have become more like personal computers, changing the trend to access the Internet. In 2011, there were nearly five billion global mobile users, and 85% of the devices they used were capable of accessing the Web. Accessing web videos, music, online gaming, social networks, e-commerce, financial services, real estate, healthcare, transportation, and many other applications through smart mobile devices has become a popular day-by-day occurrence. It is known that 61% of mobile subscribers in the United States access the Internet through their portable smart devices [4, 5, 6].

Although features and functionality of smartphones have significantly improved with time, these devices still lack processing capabilities, compared to desktops, and have limited battery life. A consequent upsurge in features and functionality of these devices has also led to greater levels of energy consumption. Such an increase in energy consumption impacts user productivity and convenience through faster depletion of the limited battery life. As a result, battery capacity and technology struggle to keep pace. Recent surveys show that the battery life of these devices is still one of the critical consumer satisfaction factors [7]. The collective energy consumed by smart portable devices is also increasing proportionally with the proliferation of their features and functionality. The total global energy consumed by smart phones is around 46.2 million megawatt hours (MWh) and that consumed by laptops is 44.6 million MWh [1]. Thus,

any advancement in energy-management techniques for these mobile devices will not only improve customer satisfaction but also help reduce the global energy consumed by these devices.

The functional capability of smart mobile devices is improving with time, but unfortunately, the life span of these devices is very short, with most users upgrading their devices every 18 to 24 months [8]. On the other hand, the processing power and mass storage of the central processing unit (CPU) doubles every one and a half years, thus driving consumers to opt for newer devices [1]. Due to their shorter life spans, millions of cell phones and other mobile devices are discarded every year. Only a small percentage of these discarded devices are recycled, and the rest contribute to electronic waste (e-waste). Unfortunately, 75% of discarded devices contribute to e-waste or landfills. In 2005, it was reported that around 2.2 million tons of e-waste were discarded, of which only 15% to 20% was recycled. The remaining 1.9 million tons ended up in landfills [9]. In 2009, millions of cell phones were sold, but only 27% of them were recycled [2]. Toxic materials present in these portable devices, such as arsenic, lead, mercury, cadmium, nickel, copper, and zinc, are capable of adversely affecting the global environment and human health [9, 10].

The challenges discussed above are some of the key driving factors for green mobile computing. Rapid growth in the field of cloud computing has become more feasible, allowing these devices to utilize the greater storage, memory, and processing capabilities of powerful remote servers. The term *cloud* is used as a metaphor for the Internet, where resources, information, software, and hardware are shared [11]. The main advantages for using cloud computing are scalability, ubiquitous availability, and maintenance costs. Today, the cloud-computing concept is being fostered by major technology companies such as Google, IBM, and Microsoft. Consider a mobile device, such as a smartphone, that is being used to play the game

of chess by a person who is mobile. The game could be played locally on the device itself, or it could be played online. In the former version, all computing required to make a move by the computer (typically the game opponent in a two-player format) is done using the device's energy resources. However, in the online version, all computations are done through a powerful remote server and conveyed through communication to the device, thus allowing the user to play games without installing a local copy and providing the flexibility of playing the game from any device, anywhere. It is easy to see that in the cloud-based scenario, more communication and possibly significantly less computation is required by the mobile device.

This dissertation explores the impact of cloud-based applications on the battery life of mobile devices. The goal of this work is to characterize those scenarios under which cloud-based applications would be relatively more energy-efficient for users of mobile devices. This dissertation also investigates factors that may affect the user experience and performance of any cloud application, as opposed to traditional non-cloud versions executed locally on a mobile device. Lastly, the issue of environmental sustainability by characterizing the overall life-cycle energy (LCE) consumed by mobile devices is addressed. This dissertation also proposes a possible solution to reduce each individual device's energy consumption (including the cost of energy to manufacture them) and the rate at which these devices are discarded. The major problems focused on in this dissertation are discussed in the following sections.

## 1.1 Energy-Efficient Operation of Mobile Devices under the Thin-Client Paradigm

With recent technological advances that improve ubiquitous connectivity and bandwidth, cloud computing has become feasible, thus allowing constrained mobile devices like smartphones, tablets, etc. to utilize the greater storage, memory, and processing capabilities of powerful remote servers. Battery life is one of the biggest constraints in mobile devices;

therefore, it becomes important to characterize the relative energy consumption of using cloud-based applications on the device, as opposed to more traditional non-cloud versions executed locally. The focus of this dissertation is to propose an energy-efficient solution for mobile devices, which have the capability of executing applications locally on the device, as well as offloading the computation to a remote cloud server. A client-server setup enables flexibility in execution of applications either on the client mobile device or on the cloud server, based on the availability of resources at each end. The theoretical limits and optimal methods for energy-efficient application management in mobile devices using cloud resources is proposed in this dissertation.

## 1.2 Quantifying User Experience for Local and Cloud Applications

The cloud-computing paradigm enables a work-anywhere-anytime scenario by allowing application execution and data storage on remote servers. The capability of accessing various local or cloud applications using smart mobile devices is also evolving rapidly. Most applications can be accessed through the Internet or by installing them locally on a mobile device. The goal of this work is to characterize those scenarios under which cloud-based applications would provide a relatively better mobile device user experience. An application-comparison metric for user experience in order to evaluate the relative functionality and performance of any applications executed on the cloud or on a local device is proposed in this work. This metric incorporates important aspects like application features, network connectivity, and relative server speedup. An algorithm to provide a tunable tradeoff between user experience and energy consumption is also designed.

## 1.3 Reducing Life-Cycle Energy Consumption for Mobile Devices

This dissertation addresses the issue of developing plausible techniques to reduce the life-cycle energy consumption of mobile devices. A green and environmentally sustainable approach by leveraging the thin-client paradigm for mobile devices is proposed. This paradigm could make progress on environmental sustainability for mobile computing on two fronts: reducing the energy consumption of each individual device (including energy costs to manufacture them) and reducing the rate at which these devices are discarded. Subsequently the life-cycle energy consumption of mobile devices is analyzed, which provides insight on where energy savings could be achieved throughout their life cycle. In this dissertation, it is argued that reduction in resource utilization on local mobile devices could be achieved under the thin-client paradigm.

## 1.4 Organization of Dissertation

The rest of this dissertation is organized as follows: Chapter 2 discusses the proposed solution for saving energy on mobile devices while executing an application locally on the device or remotely on the cloud, followed by related work, and analytical characterization and evaluation of the proposed model. Chapter 3 considers the problem of user experience when both cloud and local versions of the same application are available. This chapter also presents related work, an analysis of the application comparison metric for user experience, and the *GreenSpot* algorithm for tunable tradeoff between user experience and energy consumption. Chapter 4 presents a discussion of the issue of growing e-waste and energy consumption due to mobile devices. This chapter also presents related work, the analytical characterization of life-cycle energy consumption, and the proposed software-based thin-client solution to possibly improving the lifespan of mobile devices and reducing their utilization of resources.

# CHAPTER 2

# ENERGY-EFFICIENT OPERATION OF MOBILE DEVICES UNDER THE THIN-CLIENT PARADIGM

This chapter begins by introducing capability of a mobile device to execute an application locally on itself as well as remotely over the cloud and explaining the contributions relative to the stated problem. This is followed by a literature survey of related work in the area. Subsequently, the proposed analytical model and empirical evaluation results for the proposed model and a discussion of future work. A publication related to this work is the research of Ghose et al. [22].

## 2.1 Introduction

The primary focus of this chapter is to propose an energy-efficient solution for mobile devices. Energy consumption for any application execution is measured with respect to the mobile device locally. However, smart mobile devices have the capability of not only executing applications locally on the device but also offloading the computation to a remote cloud server. This kind of client-server setup enables flexibility in the execution of applications running them on either the client device or the cloud server—based on the availability of resources at each end while accounting for energy savings, thus providing considerable opportunity for efficient management of resources.

This chapter provides a systematic model for the power consumption of mobile device operation and uses it along with empirical measurements to study whether any potential benefits exist in terms of energy consumption at the client device end. An analytical model was designed by considering the primary power-consuming modules like display, processor, and network interface as the principal components, based on the study by Carroll and Heiser [14]. The power-

consumption model was then effectively applied to quantify the power consumed by common off-the-shelf (COTS) smartphones. Furthermore, an empirical evaluation was performed for comparing the energy consumed by a smartphone that locally executes a common application, such as word-processing, to one that offloads the application execution to a remote cloud server. Additionally, an Android-based benchmark test was developed and implemented as an application on Android smartphones. Thus, through a combination of analytical and empirical results, this work not only provides insight on the impact of various parameters on the power consumption of mobile devices, but it also establishes a future platform for developing optimal approaches for energy-efficient application management in mobile devices when remote and local domains are available. Hence, this work is expected to serve as an initial step towards designing more energy-efficient mobile devices that improve user satisfaction through longer battery life and a reduction in the environmental footprint.

The specific contributions of this work include the following:

- An analytical model for power consumption of the client for mobile devices was proposed, and necessary conditions for energy savings for application execution were developed.
- Using Android smartphones, numerical and empirical comparisons for application execution energy savings were performed to validate the proposed analytical model.
- Finally, an Android-based benchmark test was developed to implement the proposed thin-client server model. Energy savings, performance, and speedup were accounted for in the execution of the application on smartphones.

## 2.2 Related Work

Mobile devices are known to lack processing power and are deterred by their limited battery life. In order to overcome such drawbacks in mobile devices, an alternative approach

would migrate the computation to more resourceful computers (i.e., servers). Hence, offloading computations to remote servers is an attractive solution to increasing the performance of mobile devices. However, only recently has the research community started considering the energy aspects of offloading. Miettinen and Nurminen report that computation offloading can be done to improve performance and save energy, under the condition that the workload consists of more than 1,000 cycles of computation for each byte of data. Portable devices using the wireless local area network (WLAN) for remote connectivity were reported to consume less energy than third generation (3G), because the latter has more communication latencies [15]. An offloading framework for smartphones has been proposed by several researchers using partitioning and profiling of the applications. Kemp et al. [16] propose a framework (*Cuckoo*) based on the solution of a dynamic runtime system offloading using Android technology. Client-server interface modeling has been used on top of the Android framework. The developer chooses to design the interface by making use of Android Interface Definition Language (AIDL). However, Cuckoo does not support a callback mechanism and stateful services, which limits the use of this framework in a wide variety of applications requiring continuous communication between the client smartphone and the remote server (or cloud). Cuckoo framework decisions to offload the services are not based on context information, heuristics, or history, but rather on the availability of remote resources only [8].

A similar framework, called *MAUI*, was proposed by Cuervo et al. [17] for computation offloading, which considered the energy consumption in transferring or downloading the computational work or application program to a remote server. MAUI profiles applications, whereby the profiler is continuously run to determine the cost of each method. The decision for offloading is based on this criterion. A local as well as remote copy of the same application

program is always maintained on the machine. Only the program state of "remotable" procedures is offloaded. MAUI uses profiling to determine the cost of offloading each method, and it uses a linearization technique to calculate networking and CPU costs. This application is dynamic in nature because serialization is performed at runtime to determine the cost of offloading, considering the network connectivity [17]. Both Cuckoo and MAUI propose frameworks for offloading but lack a provision for appropriate guidelines in offloading existing programs. In contrast to the above-discussed frameworks, Saarinen et al. proposed the smart toolkit *Smartdiet* for offloading the computation for smartphones. The Smartdiet framework also considers the power-saving mode and traffic patterns in order to increase the accuracy of pre-existing patterns for real-time cost estimations [18].

Typically, partitioning and profiling methods of offloading techniques can be too complex and increase the run time for simple applications. This dissertation proposes a simpler, friendly, and easy-to-implement developer model rather than complex partitioning and profiling methods. Kumar and Lu [19] used an analytical model perspective to analyze the energy savings through offloading computations. Other work [17, 20] has considered a similar problem from a system-level perspective, identifying the conditions under which individual methods should be offloaded to a remote server in order to save energy. In contrast, this dissertation considers an application-level approach, as the existing applications are designed to be executed either locally or on the cloud, but not both. The work of Miettinen and Nurminen[15] aims at determining the critical factors affecting energy consumption of mobile clients in cloud computing and compares the relative impact of communication and local processing. However, their study is limited to an empirical approach without an analytical characterization. Veerecken et al. also proposes a power-consumption model for mobile devices, which defines and analyzes the power-

consumption model for thin-clients [21]. This dissertation work is based on a similar architectural model, but the focus here is only on the energy consumed by the end-client mobile device as opposed to the entire end-to-end client-server model.

## 2.3 Analytical Characterization

This section focuses on the analytical characterization of the power-consumption model for mobile computing devices under a thin-client paradigm with a formal problem definition. Subsequently, energy-savings conditions under the thin-client paradigm are also derived. The variables used in the model are defined in Table 2.1.

TABLE 2.1

VARIABLES DEFINED

| Variable | Definition |
|---|---|
| $P_o$ | Base power of mobile device |
| $\delta$ | Load on mobile device |
| $P_{CPU}$ | Power consumed by device CPU |
| $P_{CPU}^{idle}$ | Power consumed by mobile CPU idle state |
| $P_{CPU}^{act}$ | Power consumed by mobile CPU active state |
| $P_{Display}$ | Power consumed by mobile display |
| $P_{Dis}^{idle}$ | Power consumed by mobile display idle state |
| $P_{Dis}^{act}$ | Power consumed by mobile display active state |
| $P_{NIC}$ | Power consumed by mobile NIC card |
| $P_{NIC}^{idle}$ | Power consumed by mobile NIC card in idle state |
| $P_{NIC}^{act}$ | Power consumed by mobile NIC card in active state |
| $P_{NIC}^{sleep}$ | Power consumed by mobile NIC card in sleep state |
| $P_{tot}$ | Total power consumed by mobile device |
| $P_{tot}^{L}$ | Total power consumed for executing local application |
| $t_L$ | Total time consumed for executing local application |
| $E_L$ | Total energy consumed for executing local application |
| $P_{tot}^{R}$ | Total power consumed to execute application on remote server |
| $t_R$ | Total time consumed to execute application on remote server |
| $E_R$ | Total energy consumed to execute application on remote server |

### 2.3.1 Power Consumption Model

Smart mobile devices have the capability of executing applications locally on a device as well as offloading the computation to a remote cloud server and then have the result displayed on the client device. Let the total power consumption of the device ($P_{tot}$) be separated into four major components: power consumed by the processing unit ($P_{CPU}$), power consumed by the display ($P_{Display}$), power consumed by the network interface card of the device ($P_{NIC}$), and remaining power, which is considered the base power of the device ($P_o$). Therefore, the total power consumption of the mobile device to execute any task/ application is defined as

$$P_{tot} = P_o + P_{CPU} + P_{NIC} + P_{Display} \tag{2.1}$$

In this model, it was primarily assumed that all computations required by the application would be executed by the processor. Two chief states of the processing unit, namely *active* and *idle*, were the primary contributors for power consumed by the processing component of this model. It was also assumed that the power consumed by the processing unit was in the active state, that is, during the processing of a task as a function of the load δ. A linear dependence on the load was assumed for power consumed by the processor. Load on the CPU could vary between 0 and 1 for no-load and full-load conditions, respectively. The processing unit could remain in either active or idle states (or $f_{CPU}^{idle} + f_{CPU}^{act} = 1$). Therefore, the total power consumed by the processing unit is defined as

$$P_{CPU} = (1 - f_{CPU}^{act}) P_{CPU}^{idle} + \delta f_{CPU}^{act} P_{CPU}^{act} \tag{2.2}$$

A significant amount of power was considered to be consumed by the liquid crystal display (LCD) or organic light-emitting diode (OLED) display of the device. In this model, the power consumed by the display was estimated as a linear function of the brightness constant $\beta$,

which can vary between 1 and 255. The power consumed by the display can be in either idle or active states. Thus, the total power consumed by the display is defined as

$$P_{Display} = \beta((1 - f_{Dis}^{act})P_{Dis}^{idle} + f_{Dis}^{act} P_{Dis}^{act}) \quad (2.3)$$

The power consumed by the network interface card (NIC) was also considered a function of its three different states: active, idle, and sleep. The interface card was assumed to be in the active state while transmitting and receiving data. When the wireless interface was not in use, it was considered to be in sleep mode, and for the rest of the time it was regarded as idle. The NIC was assumed to be in each state for a fraction of time $f_{NIC}^{(.)}$, where $f_{NIC}^{idle} = (1 - f_{NIC}^{act} - f_{NIC}^{sleep})$. The collective power consumed by the NIC of the mobile device $P_{NIC}$ is defined as

$$P_{NIC} = f_{NIC}^{idle} P_{NIC}^{idle} + f_{NIC}^{act} P_{NIC}^{act} + f_{NIC}^{sleep} P_{NIC}^{sleep} \quad (2.4)$$

### 2.3.2 Energy-Saving Condition

The conditions necessary for saving energy on a mobile device by means of the thin-client paradigm are discussed in this section. It was assumed (for a fair comparison) that for those computations performed locally (on the device), the communication interface would not need to be active. To ensure that applications were executed locally, they could be pre-installed on the mobile device. In this case, no communication between the client and the server could be presumed. All computations required by the application were executed by the processor on the local device. In the case of remote computations or thin-client-server paradigm, access to the application was provided through remote cloud servers using the Wi-Fi interface of the device. Therefore, a necessary condition for an execution to be energy efficient under the thin-client paradigm was that the energy consumed for remote execution be less than the local execution energy, as defined in equation (2.5).

$$Energy\ Ratio = \frac{E_R}{E_L} < 1 \tag{2.5}$$

Next, in equation (2.6), the energy consumed for local execution of the application $E_L$ is characterized as the product of power consumed by the CPU and the time required to process the task. The Wi-Fi interface is considered idle in this scenario.

$$E_L = P_{tot}^L * t_L \tag{2.6}$$

Energy consumption of the mobile device when the task is executed remotely ($E_R$) is shown in equation (2.7). Energy consumption by the local mobile device is linearly dependent on the power consumed by the total power consumed by the local mobile device and total end-to-end time for remote processing ($t_R$).

$$E_R = P_{tot}^R * t_R \tag{2.7}$$

Thus, the total end-to-end time for remote execution includes the time consumed by the client device to send and receive data from the remote server ($t_{NIC}^R$), and the time required by the remote server to complete the task ($t_{CPU}^R$):

$$t_R = t_{CPU}^R + t_{NIC}^R \tag{2.8}$$

The time required to transfer or offload all the data to a remote server for computation is equal to twice the one-way network latency (one-way trip time or OTT) between the client (local mobile device) and remote server, and defined as

$$t_{NIC}^R = 2 * \text{OTT} \tag{2.9}$$

It is assumed that the remote server execution time is *n* times faster than the local mobile device. Hence, speedup can be defined as

$$Speedup = \frac{t_L}{t_R} = n \tag{2.10}$$

Thus, the final energy ratio can be defined as

$$Energy\ Ratio = \frac{1}{n}\frac{P_{tot}^{R}}{P_{tot}^{L}} < 1,\ where\ \frac{P_{tot}^{R}}{P_{tot}^{L}} = Power\ Ratio \quad (2.11)$$

The energy ratio is directly proportional to the total power consumed by the local device for remote execution. However, it is inversely proportional to the speedup and the total power consumed by the local device for local execution of the same application.

## 2.4 Numerical Evaluation of Power Consumption Model

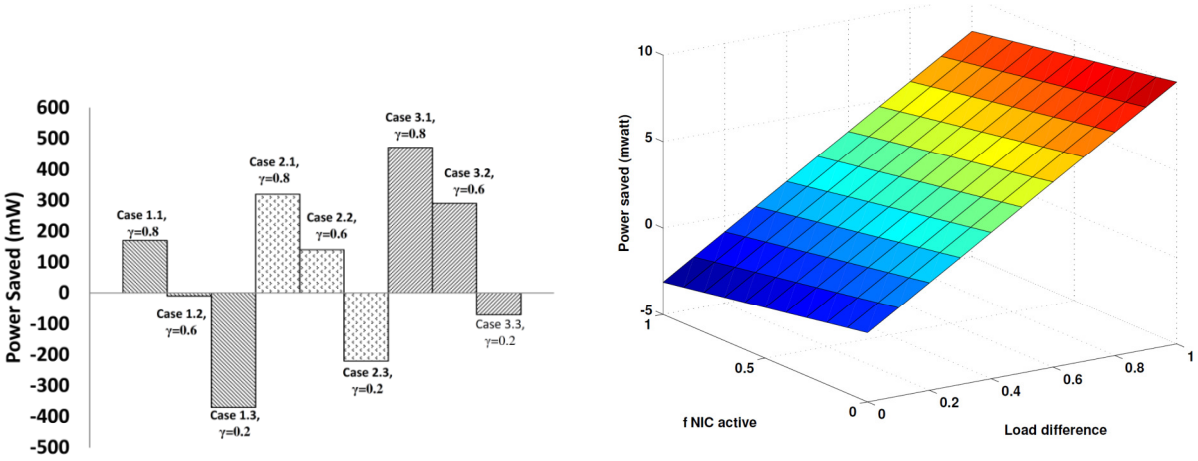
In this section, a preliminary numerical evaluation of the power consumption trends on a smartphone mobile device is presented. This preliminary evaluation serves to validate the proposed analytical model discussed in section 2.3, to explore other factors that may influence power consumption, and help understand and further explore the energy-saving condition proposed in equation (2.5) for mobile devices working under a client-server paradigm. The following assumptions were considered in order to justifiably evaluate the power savings:

- For the computations to be performed locally on the device, the communication interface does not need to be active.
- In order to ensure that applications are executed locally, they can be pre-installed on the mobile device; therefore, no communication between the client and the server can be presumed.
- In the case of local execution, all computations required by the application will be executed by the processor on the local device.
- In the case of remote computations or a thin-client-server paradigm, access to the application is provided through remote cloud servers using the Wi-Fi interface of the device.

- The base power and power consumed by the display remain constant for both cases, i.e., when the application is executed locally versus the same application executed remotely under the thin-client paradigm.

The condition stated in equation (2.11) was investigated by considering various combinations of operational states for the network interface card for small form-factor devices. The speedup ($n$) was assumed to be equal to one for this analysis. It was also assumed that the base power ($P_o$) and power consumed by display ($P_{Display}$) remain constant for both cases, i.e. when the application is executed locally and when the application is executed remotely under the thin-client paradigm. In addition, for the computations performed locally on the device, there is no requirement for the communication interface to be active (i.e. $(P_{NIC})_{local} = 0$)). In this analysis, it was considered that the communication interface could only be active or idle, thus $f_{NIC}^{idle} + f_{NIC}^{act} = 1$, while executing the task remotely under the thin-client paradigm. The difference in load on the client device between local and cloud-based processing is equal to $\gamma$. The load difference $\gamma$ and the fraction of time for NIC states were varied to theoretically study the impact of communication and computation on power consumption. In each case, the fraction of the time network interface states was kept constant, and the load difference on the client was varied, as shown in Figure 2.1(a). Here, Case 1 considers that the client utilizes the NIC interface actively; hence, $f_{NIC}^{act}$ = 0.9, $f_{NIC}^{idle}$= 0.1. Case 2 represents applications that do not require frequent communication between the remote cloud server and the client; therefore, $f_{NIC}^{act}$ = 0.6, $f_{NIC}^{idle}$= 0.4. Case 3 considers the scenario where the wirelesses interface of the mobile device is in idle mode for most of the time: $f_{NIC}^{act}$ = 0.3, $f_{NIC}^{idle}$= 0.7. Figure 2.1 (b) shows the power savings when the load difference and $f_{NIC}^{act}$ is varying. The measured power-consumption values for a

smartphone, from the work of Carroll and Heiser [14], were used for this numerical evaluation: $P_{CPU}$= 900 mW, $P_{NIC}^{act}$ = 600 mW, and $P_{NIC}^{idle}$ = 100 mW.




(a) Varying load difference and f(.) NIC (b) Varying load difference and $f_{NIC}^{act}$

Figure 2.1: Power saved for various cases of operating conditions.
(note that for some cases, negative power is saved, that is, more power is consumed by offloading tasks to a remote server).

It can be seen in Figure 2.1(a) and (b) that power can be saved on mobile devices if the load difference is greater than the power consumed by the wireless interface to process and send the data for remote processing. This suggests that if the application requires less computation, it is energy efficient to handle the task locally. Similarly, for applications that require more complex computation, remote processing would be energy efficient. To further validate this model and the numerical evaluation results, empirical evaluations were performed and are discussed in the next subsection.

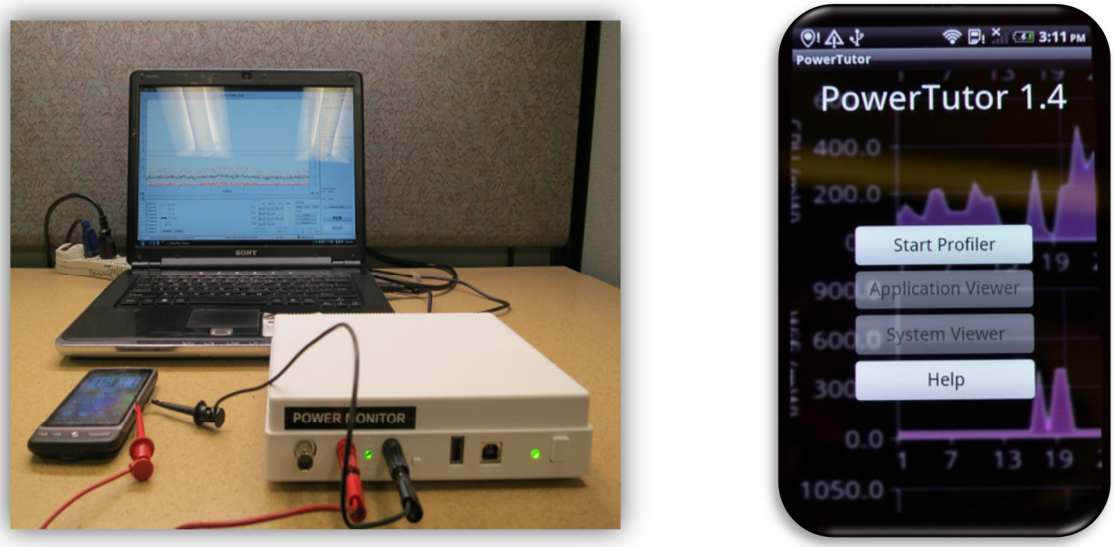
## 2.5 Empirical Evaluation of Power Consumption

In this subsection, an empirical comparison of power consumption trend for local and remote application execution on smartphones is presented. All evaluations were executed on common off-the-shelf smartphones. Measurements were performed on two different Android-

based smartphones—HTC desire and Motorola Milestone XT 720—and aggregated across multiple runs.

### 2.5.1 Power Measurement Methodology

In order to provide a methodical comparison of power consumption for remote and local applications, a set of chosen applications was executed in each test measurement after installing them on a local device as well as at a remote location on the cloud. The corresponding power consumptions in each case were measured using the Monsoon power meter, as shown in Figure 2.2(a) In order to measure the power consumption on the mobile device, a commercially available application, namely *PowerTutor*, was used, as shown in Figure 2.2(b).




(a) Monsoon power meter lab setup (b) PowerTutor application

Figure 2.2: Power measurement methodology.

The Monsoon power meter logs power usage for every millisecond. A subtractive method was used to calculate the total power consumption by the specific application under the test [23]. PowerTutor logs the power usage of each hardware component during the processing of applications. It was reported that PowerTutor's long-time average error is less than 2.5% [24].

Power consumption was logged for each second, and each application was mapped with a unique user identification (UID) number [25]. An automated program was developed in Java to calculate the total power consumption using the power log from PowerTutor. The program was coded with the regular expression matching the pattern. Power consumption for total executions of the local program was calculated by summing the value of the CPU power consumption ($P_{CPU}$) for that process, and power consumption for the remote case was calculated by adding the total CPU power ($P_{CPU}$) with the Wi-Fi power ($P_{NIC}$) for each application.

### 2.5.2 Experimental Methodology

Typical experiments consisted of search operations performed on Word files of different sizes to locate a specific set of words. The executions on the local device were performed through *Quickoffice*, a word processing suite available on Android devices. *Google Documents* (hereafter referred to as *Google Docs*), a cloud-based word processing suite that allows users to create, edit, and save documents on remote servers, all in real-time, was used to test the equivalent cloud-based application. During the experiments, all synchronizing services on the device were disabled. The Wi-Fi and 3G interfaces were turned off while executing the applications locally. The Wi-Fi interface was the only external-access application that was left on while executing the search operation remotely. During the experiment, the following steps were considered in order to justifiably evaluate the power savings.

- For the computations to be performed locally on the device, the WiFI communication interface was turned off.
- In order to ensure that applications were executed locally, they were pre-installed on the mobile device.

- In the case of local execution, all computations required by the application were executed by the processor on the local device.
- In the case of remote computations or thin-client-server paradigm, access to the application was provided through remote cloud servers using the device's Wi-Fi interface.
- The base power and power consumed by the display remained constant for both cases, i.e., when the application was executed locally and when the same application was executed remotely under the thin-client paradigm

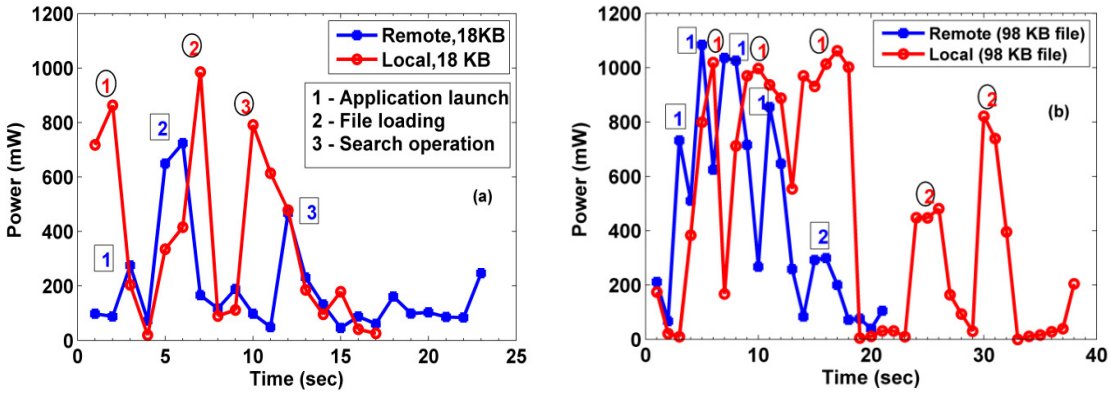
### 2.5.3 Power Consumption Evaluation: Local versus Remote Execution

It was observed during the experiments that the OLED consumed almost the same amount of power ($P_{Display}$ = approximately 520 mW) when the application was executed either locally or remotely, which is consistent with the assumption discussed in the previous section. Therefore, the subsequent analysis includes the total power consumed by the smartphone, excluding the power consumed by the display ($P_{Display}$). In order to test the tradeoff between computation and communication, it was challenging to find an existing open-source application. In order to provide a systematic comparison between the local and remote execution, several text files of different sizes were created, and a search for a specific set of words in the complete file was executed. In the case of local execution, these text files in kilobytes (kB) were saved locally on the device, and for remote execution, the same documents were created on Google Docs. As a check for the consistency of the experiments, the executions were repeated five times, and the average of the power consumed was computed during the analysis.

Power consumed during the execution of either the local or remote application was computed as a cumulative of the total power consumed ($P_{tot}$) by all components of the device for each millisecond. A comparison of the power consumption versus time in seconds for the

locally executed and remotely executed applications for the two text file sizes of 18 kB and 98 kB are shown in Figures 2.3(a) and 2.3(b), respectively. As shown, power consumption ($P_{tot} - P_{Display}$) peaks observed at various time stamps refer to the different stages of the search operation, as indicated with numbers 1 to 3 on the plot.

In the case of a locally executed application—red line in Figure. 2.3(a)—the peak numbered 1 indicates the total power ( $P_{tot} - P_{Display}$) consumed when an application such as Quickoffice is launched on the local device. The peak numbered 2 indicates the total power consumed when the text file is opened/accessed through the previously launched Quickoffice application. Similarly, the peak numbered 3 occurs as the maximum power consumed when the search operation is executed. It can also be observed from Figure 2.3(a) that the total power consumed reaches a minimum between events as the processor goes into a near-idle state until the next event is triggered by the user.




(a) (b)

Figure 2.3: Power consumption ($P_{tot} - P_{Display}$) profiles for search operation performed on (a) 18 KB and (b) 98 KB Word files.

The solid curve in Figure 2.3(a) shows the power consumption profile when the search operation was performed on a remotely stored text file. The power consumption stages included in this task were the launching of the browser (including access to Google Docs) on the local

device as indicated by peak 1, loading the text file on to the word processing application (peak 2), and execution of the search operation (peak 3). An important observation from these plots is that the power consumed by the local device in the case of a remotely executed application task is always lower than that of the locally executed counterpart. This indicates that a certain percentage of power is always saved when the computational load is offloaded to the remote server.

Testing on the 18 kB text file was followed by the implementation of the same experimental procedure involving the same set of search operations on a 98 kB file. The resulting power consumption profile is depicted in Figure 2.3(b). This test included two operational steps because the application to access the text file was already opened in the first phase (18 kB file) of the testing. As can be seen from Figure 2.3(b), local execution of the tasks (red curve) included two sets of peaks indicated by numbers 1 and 2. Peak 1 shows the power consumed during the loading operation of the text file on to the already-running Quickoffice application. In contrast to the tests on the 18 kB file, the loading of the text file took nearly 14 seconds due to the larger size of the chosen text file. The loading of the file ended at approximately 17 seconds on the time scale. The next set of peaks, numbered 2, indicate the total power consumed when the search operation was performed. On the other hand, when the search operation was performed on the remote cloud, the loading of the text file into the already-running Google Docs application was completed in approximately 12 seconds. However, during the search operation, as marked by the peak numbered 2, the total power consumed was approximately 50% lower than that of the maximum power utilized when the same operation was performed locally (peak numbered 2 on the red curve). Analogous to the results from testing on the 18 KB file, these curves also show

minimum power usage points between tasks where ideally the processing unit goes into the idle state and effectively very little power is consumed.

The imperative observation through the above experiment is that the total time to complete the computation locally and remotely differs significantly for a larger-size file, which in turn affects the drainage of battery energy from the client mobile device. Therefore, analyzing the energy savings is also equally important, which is studied in the next section.

## 2.6 Evaluation of Possible Energy Savings

In this section, an empirical comparison of total energy consumption and energy savings for both local and remote word processing application execution on smartphones is presented. All evaluations were executed on COTS smartphones. The measurements were performed on two different Android-based smartphones—HTC desire and Motorola Milestone XT 720—aggregated across multiple runs.

### 2.6.1 Total Energy Consumption: Local versus Remote Execution

Figure 2.4 depicts the comparison of energy consumption for a varying number of words in the text file being tested. The maxima of the bars represent the mean of aggregate energy consumed by the device for execution of applications over five runs. The error bars represent the standard deviation over each test run. As shown, the increasing trend in energy consumption in the case of locally executed applications indicates that the total energy consumption on the local device is a function of the size of the text file. An exponential increase in the consumed energy can be observed as a result of increase in the file size. An approximately ten-fold increase in total energy consumption was observed when the number of words increased from 303 (18 kB) to 72,102 (219 kB) during local computations. Relative to the local computations, the remotely executed computations did not consume much power, even when the file size was increased. The

increase in energy consumption was comparatively lower, with an approximately four-fold escalation, with the same increase in the number of words from 303 (18 kB) to 72,102 (219 kB). This result, shown in Figure 2.4, supports the analytical model defined earlier by equations (2.5), (2.6), and (2.7) in section 2.3.

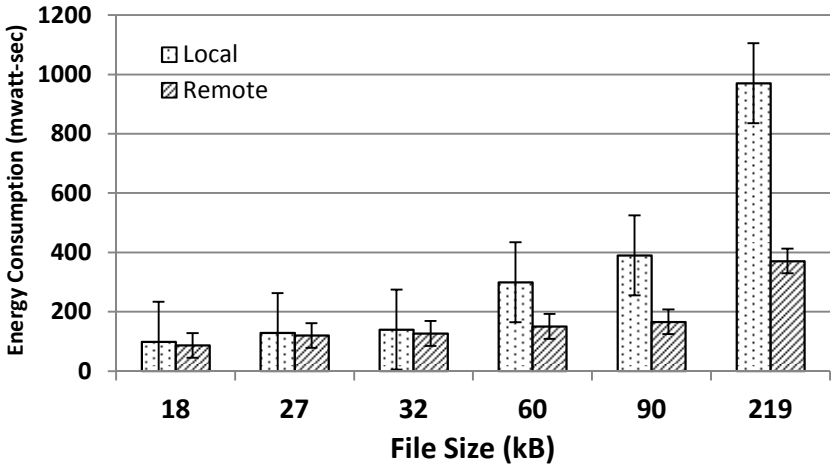



Figure 2.4: Comparison of aggregate energy consumption for local and remote execution of search operation.

Hence, it can be inferred that it is more energy efficient to execute the applications on the cloud remotely when the task requires more computation and local resources. Thus, remote execution can indeed save energy for such computation-intensive applications.

### 2.6.2 Possible Energy Savings through Offloading

Figure 2.5 shows the energy savings obtained while executing the search operation under the client-server paradigm on different files of varying sizes. As shown, the energy savings is not continually proportional to the size of computation. It can be seen that the energy consumed for processing the search operation on a file of size 27 kB was larger than the required for processing the same operation on a file of size 32 kB. The variation in energy savings arises from the fact that it also depends on network conditions and time required to access the file from the remote server. From the power logs, it was determined that power consumed by the wireless

medium also plays an important role in the process of evaluating the energy-efficient trade-off between wireless communication for remote processing and local computation. The notable fact is that there is still some power savings for cloud applications, which validates the condition proposed in equation (2.5).

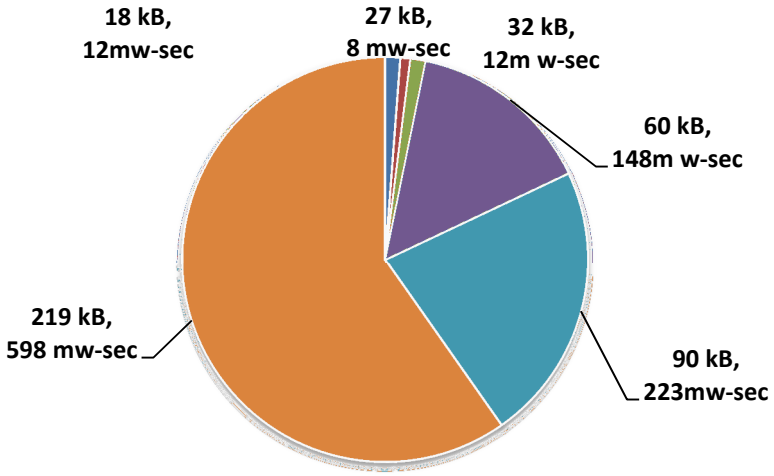



Figure 2.5: Energy savings for remote execution of various sizes of text files.

## 2.7 Thin-Client Benchmark Application Development Framework using Android Architecture

The following section describes the general Android application architecture. The test applications and benchmark developed in this dissertation were performed using the Android software development kit (SDK) tool. The benchmark test for the thin-client model was tested and executed on Android devices for this dissertation. Therefore, this section explains the basic architecture of any Android-based application.

### 2.7.1 Android Application Architechture

Android applications were developed in the Java programming language using the Android software development kit (SDK) tool. This tool compiles the code as well as data and resource files present in each project into an Android package. The complied Android package is an archive file with an ".apk" suffix and is used to install the Android application on Android-

powered devices. The front-end and back-end components for both local and remote applications were implemented using the activity and service mangers present in the Android application architecture. In this software, XML acts as the front-end language and Java acts as the back-end language. The main content for this application was developed in the Java back-end language. In the front-end text views, text fields and buttons were included in the XML page. Various components of the application, such as *activity* and *services*, were activated using an asynchronous message called *intent*. Intent binds all the components of the application during the run time. The *apk* file was installed on the client device (HTC Desire smartphone) to execute and test the applications [22, 27, 28].

The activity component present in the application serves as a visual representation of an application, i.e., window for user interaction (UI) to input the random numbers for the gaming application. General activity and service component life cycle in any applications is explained in the following subsections.

### 2.7.2 Activity Component Life Cycle

Figure 2.6 presents the activity component life cycle for any android application. The activity is first created using the *OnCreate* procedure. This involves a Linux procedure creation and memory allotment for all UI objects. The *Activity Manager* manages the complete life cycle of any activity. It creates the activity when the user starts the application. Resources are acquired in the OnCreate procedure and released on the *OnDestroy* procedure. The Activity Manager keeps switching the activity to the background and foreground, depending on the choice of application user. The Java Garbage Collector Activity Manager destroys those activities that are not used by the user. This garbage collection process helps to quickly start the activity in the background. The user can interact with the activity between *OnStart* and *OnStop. OnResume* and

*OnPause* are the method calls between which the activity is on the foreground and ready for user interaction. The activity object is maintained in memory even after the activity is paused or stopped.

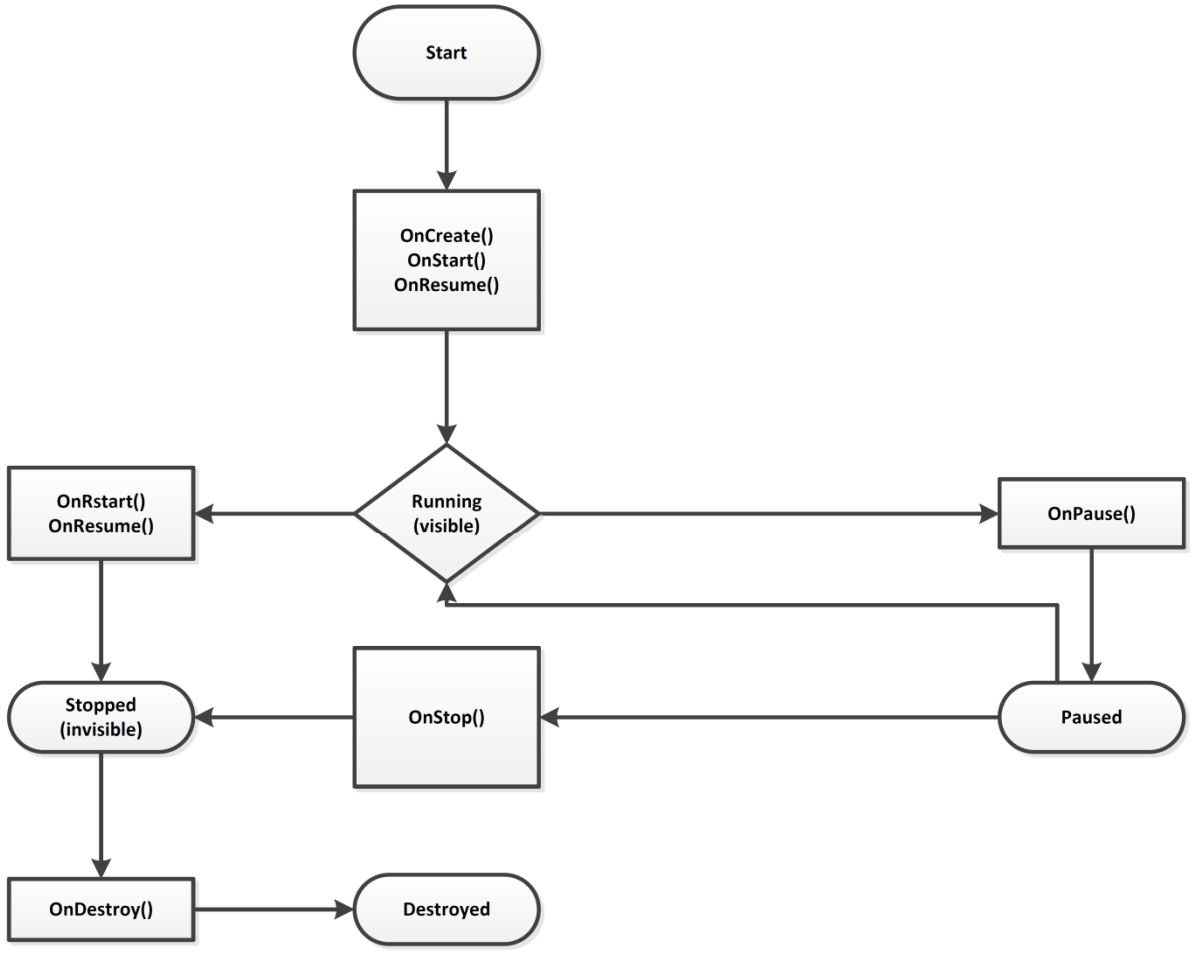



Figure 2.6: Activity life cycle from its creation until its end [28].

### 2.7.3 Services Component of Life Cycle

Services are used to support the activity component of the application. It does not possess any user interface; it can run in the background even after the user switches to another

application. The application activity can bind to a service and perform the Interprocess Communication (IPC) to delegate some non-UI tasks to it, for example, running music in the background, handling network connections, reading or writing back to a file, and many more.

Services acquire resources in OnCreate and relinquish them when OnDestroy is the method called, as shown in Figure 2.7. Both the *startService* and *bindService* methods call OnCreate and OnDestroy, respectively. Services should be designed carefully by the developers and should be called appropriately, so that the CPU cycles are not wasted and the critical resources like memory, battery, CPU, etc. are efficiently utilized.

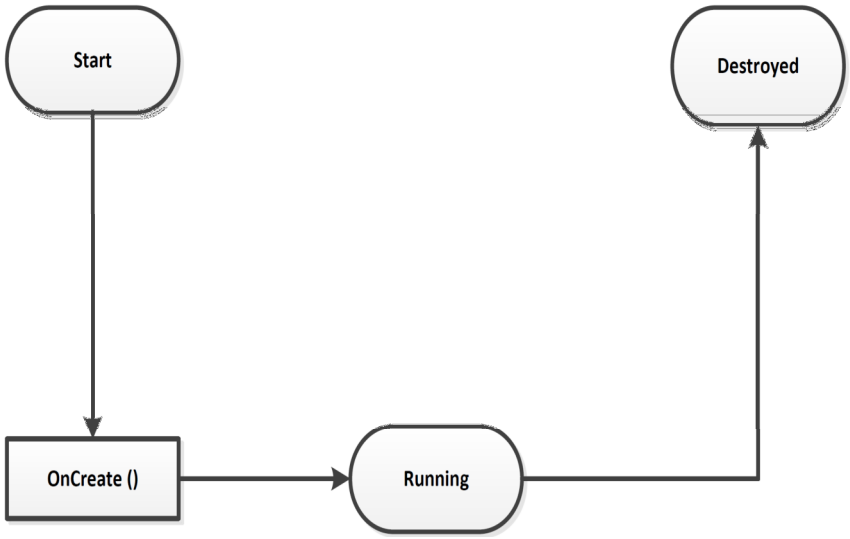



Figure 2.7: Service life cycle from its creation until its end [28].

## 2.8 Laboratory Test-Bed Setup for Thin-Client Server Model

This section, presents the implementation of a thin-client server model for smartphone devices based on empirical and analytical results. The proposed computation offloading model was implemented and tested on an Android smartphone. The application begins with user input to execute computation for a specific task on the mobile device. The thin-client server model for smartphones was implemented using a socket program. For this experiment, a Dell laptop

equipped with Intel® Core™ i3 CPU (2.53 GHz) was used as the server, while an Android HTC-Desire smartphone was implemented as the client.

A certain dearth in the compatible open-source applications for benchmarking testing makes it challenging to test the proposed thin-client server architecture for the prevailing applications. Hence, a gaming application called "Lucky Numbers," as shown in Figure 2.8, was developed for the benchmark testing. In this application, the user wins if two numbers (input by the user) are found in a file consisting of a preset random numbers. Files with different quantities of random numbers (sample sizes) are considered to attain better accuracy of the results.

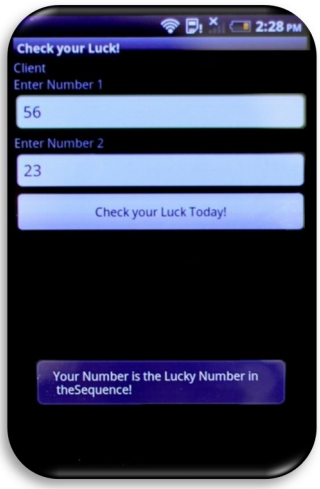



Figure 2.8: Benchmark test application layout.

These test files consisting of random numbers were stored in an asset folder separately on the local device as well as the remote server for access during application logic executions. In order to develop the gaming application, *Eclipse IDE*, was utilized. The Android Development Tool (ADT) plug-in was also installed on the Android 2.2 platform for development of these test applications.

### 2.8.1 Application Framework for Local Application Execution

The application logic for local application execution on the client device was developed in order to execute the logic on the smartphone. This program utilizes the CPU and other resources

of the smartphone for the computation. A simple graphical user interface (GUI) embedded with a button labeled “Local” and the required fields to enter the user input were also developed using Java and XML, as explained in the previous subsection. An event was generated for each click of the button, followed by the extraction of the input file in the asset folder. The set of input numbers to be searched was entered by the user via the GUI.

### 2.8.2 Application Framework for Remote Application Execution

In a remote application framework, the program is initiated on the client device, i.e., the smartphone, but the computation is executed on the remote server. The remote execution client and server scenario was implemented using socket programming, as shown by the design in Figure 2.9. The Android smartphone was installed to act as the client. In this experiment, the remote server was a Dell laptop equipped with an Intel® Core™ i3 CPU, and the client was an Android HTC Desire smartphone. The server processor frequency was 2.53 GHz, and the smartphone processor frequency was 1 GHz. A GUI interface similar to that of the local execution scenario was developed and incorporated with a button labeled “Remote.” The user is expected to enter the random numbers in the available fields. The event is generated after the user enters the input for the application, and the button labeled remote in the GUI is clicked. The input file extracted from the asset folder is directed to the remote server as a byte stream to the *ObjectInputStream* in the Java socket interface. After completion of the requested task by the client, the sever sends the results back to the client. The final result is displayed on the client’s device (smartphone).

## 2.9 Analysis of Resource Utilization

The primary focus of the empirical testing was to identify benefits of the cloud-based thin-client approach for mobile devices over traditional mobile devices where all computation is

executed locally on the device. Therefore, the power consumption for each hardware component was measured, and energy savings for each sample was analyzed, as will be discussed later in this section.

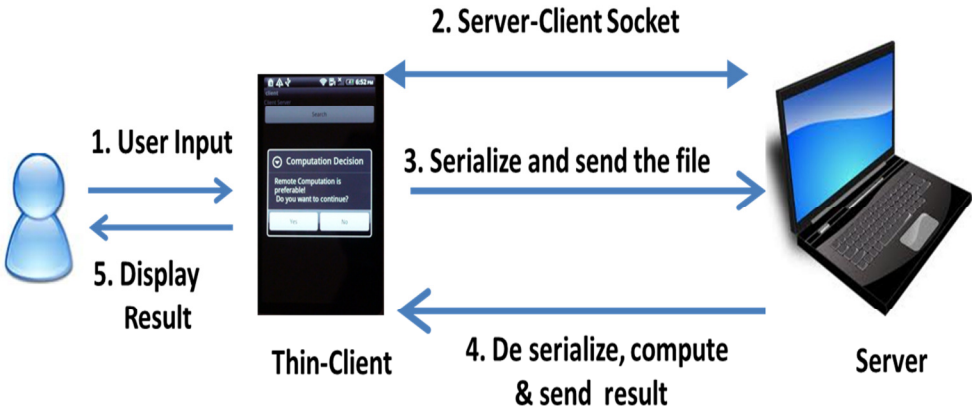



Figure 2.9: Remote client server paradigm for executing applications.

### 2.9.1 End-to-End Computation Time Measurement

The end-to-end computation time was logged using the *System.currentTimeMillis* method in Java. This method returns the total computation time consumed in milliseconds. In the case of application execution on the local device, time was logged from the instant of user input (event generation for computation) until display of the final result. However, in the case of the thin-client paradigm where the application logic is executed on a remote server, the time was logged from the instant of the user input on the client device until the result is generated and sent back to a calling program on the client (smartphone). The intermediate step in this process includes the time taken to send the file as an object stream to the remote server (laptop).

By means of empirical testing, it was found that the utilization of CPU time for computation increases exponentially for larger computations on the local smartphone; however, under the thin-client paradigm (remote computation), the CPU utilization was observed to remain approximately constant, as shown in Figure 2.10. In the thin-client scenario, a low CPU

time utilization is the consequence of a comparatively fewer number of computations executed by the local smartphone. The CPU, in this case, is only utilized for sending the input data to the remote server and for subsequently displaying the final results. Approximately 108 times increase in the computation time was observed when the number of random numbers increased from 10 to 20,000 during local computations.

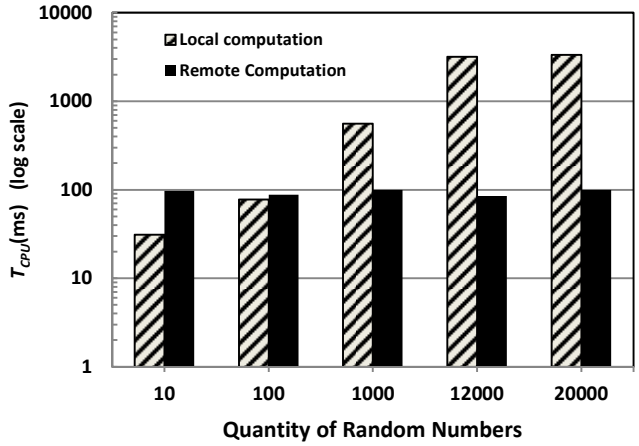



Figure 2.10: Computation time consumption versus quantity of random numbers in each file for local and remote application execution in smartphones.

### 2.9.2 Energy Savings

In order to calculate the power consumption, a similar methodology as explained in section 2.5.1 was used. The percentage of energy savings for execution of applications under the thin-client-server paradigm is defined as

$$E_{savings}\ (\%) = \frac{E_{t_{local}} - E_{t_{remote}}}{E_{t_{local}}} \times 100 \tag{2.12}$$

The $E_{savings}$ for the first test case with ten random numbers was found to be negative (excess energy consumption). This excess consumption is indicative of the energy required to process and send the file for remote computation ($E_{t_{remote}}$) being higher than the energy required for local computation ($E_{t_{local}}$). Based on power log analysis, power consumption by the wireless interface of the device ($P_{Wi-Fi}$) contributes to a larger portion of the total power ($P_{t_{remote}}$).

It can be observed from Figure 2.11 that energy could be saved on mobile devices under thin-client scenario if the processing energy consumed on the local device is greater than the energy consumed by the wireless interface to process and send the data for remote processing.

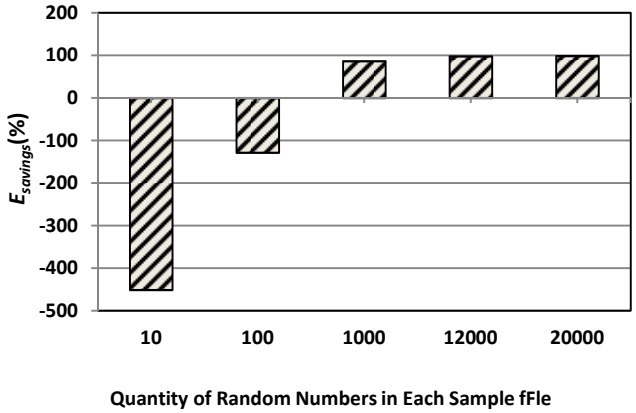



Figure 2.11: Energy saved (%) on smartphones as a function of size of test file for application execution under thin-client-server paradigm.

It was also observed from the energy logs that for the test case with 20 K random numbers, the computation energy for local computation was approximately 71 times more than that of the thin-client-server paradigm. This suggests that if the application requires less computation, it is more energy-efficient to handle such a task locally. Similarly, for applications that require more complex computation, remote processing would be energy efficient for small form-factor devices.

## 2.10 Conclusion and Future Work

A power consumption assessment for mobile smartphones working under the remote client-server paradigm was presented. Conditions that can lead to a reduction in power consumption by a mobile device through offloading of tasks were discussed. Using a remote cloud-based approach, the energy savings was calculated. Finally, a proposed mobile device computation offloading model was implemented by developing an application on Android smartphones to determine when to offload application tasks to a remote server in order to save

energy. The results of this work not only provide insight on the impact of various parameters on the power consumption of mobile devices, but they also provide future guidance to develop optimal approaches for energy-efficient application management in mobile devices when remote and local domains are available.

In this work, power consumption of the local mobile device increases with utilization of local resources like CPU and the network interface is analyzed. In order to reduce the power consumption and local resource utilization, computation-intensive applications could be executed on remote cloud servers. With the option of execution of any application on remote cloud servers there exists variability in the computation load on the local mobile device CPU. In order to achieve maximum performance for a given power limit, a control system-based model could be developed as future work. Furthermore, an optimal approach could be used for a predictive system. The predictor module in the system would assist the mobile device in predicting the system configuration so that the device could provide maximum performance within the given power limit by choosing either local or cloud resources. The predictor module could be defined based on power, performance, and the user experience model proposed in this dissertation.

# CHAPTER 3

# QUANTIFYING USER EXPERIENCE ON MOBILE DEVICE FOR LOCAL AND CLOUD APPLICATIONS

This chapter begins by introducing the differences and capabilities of mobile-device cloud and non-cloud applications that may affect the user experience. This is followed by a literature survey of related work in the area. The subsequently proposed analytical model and proposed algorithm are then presented. Implementation of the proposed algorithm and future work are discussed. In addition to publications under review, two of them related to this work have been published [36, 37].

## 3.1 Introduction

The capability to access various cloud-based applications using smart mobile devices is evolving rapidly. Many of these applications could be accessed via the Internet or by installing them locally on a mobile device. Applications that could be installed locally on a mobile device are designed specifically to run on the device's operating system, and they are typically customized to fit the device's screen size. However, with access to the Internet, cloud-based applications could be accessed from any mobile device. The Internet or data connection speed of connectivity for these devices impacts the user experience. A few applications could be accessed over a web browser, but they require no further network involvement and are executed locally (e.g., Adobe Flash-based games). For instance, the amount of processing required for playing games locally depends on the nature of the game. Flash-based games require little local processing compared to games like chess that can require extensive processing from the "computer" opponent. Moreover, games played over the network, possibly against other players

or against a computer, can vary in the amount of communication required, based on the interactivity of the game and with little local processing required [24].

Different types of user experiences with both local and cloud-based web applications exist. Every application varies in the degree of local computation and the required network communication. Typical word processing applications that run locally on the device have the advantage of requiring no network connectivity and a rich feature set on the software. However, a cloud-based version of word processing provides ubiquitous access through any terminal device at any time through a network connection. Local execution of word processing software is expected to involve little or no communication but could use a good deal of resources to run the application. On the other hand, cloud-based execution of word processing software would require some communication but little local processing. This dissertation describes an approach to quantify the user experience for those available local and cloud-based versions of applications.

The expected contributions of this chapter include the following**:**

- An application comparison metric for user experience is developed to evaluate the relative functionality and performance of any applications executed on the cloud or on a local device. This metric incorporates important aspects like application features, network connectivity, and relative server speedup.
- A numerical evaluation of the user-experience metric, *AppScore*, is performed to validate the proposed analytical model.
- An algorithm, GreenSpot, is developed and can be deployed on mobile devices to seamlessly decide between cloud or non-cloud versions of applications. This algorithm considers the AppScore rating of applications if executed over the cloud or locally, and allows a tunable tradeoff between user experience and energy savings.

- GreenSpot is implemented as an "app" for Android operating system (OS)-based smartphones.

## 3.2 Related Work

Most prior work related to mobile applications and the cloud has been done with respect to quality of service (QoS). Some of the recent academic and industrial research work has focused on the area of quality of experience or user experience for mobile applications and cloud services. Wac et al. analyzed different factors that influence the quality of experience. They also reported that QoS is a critical user-experience factor for highly interactive applications [29]. Huang et al. [30] evaluated the QoS metric based on the download speed of the application, speed of the domain name server (DNS), and transport control protocol (TCP) three-way handshake. The work of Dinda et al. [31], who studied application performance and user experience for client-server mobile applications, is comparable to the research described in this dissertation and related publications. Their work evaluated the relationship between the user experience and application performance for desktop applications, whereas the work here proposes an app score to evaluate the relative functionality and performance of any applications executed on the cloud or executed locally on a mobile device.

Migrating the computation from a mobile device to more powerful cloud servers has been examined for many years now (e.g., [32]), mainly for performance gains. It is only recently that such work has considered the aspect of energy consumption of mobile devices [17, 33–35]. Kumar and Lu [19] used a simple analytical model to determine whether an offloading computation could save energy. Although a useful start, this work did not study application characteristics, empirically or analytically, and the model used did not consider aspects like the power save mode (PSM) of the wireless interface or performance constraints imposed by users.

The work of Cuervo et al. [17] and Chen et al. [34] approach the problem from a system-level perspective, identifying opportunities when code and its individual methods should be offloaded to a server in order to save energy. In contrast, the work in this dissertation takes an application-level perspective and considers an application as a single unit that cannot be decomposed into multiple methods and must be run either on the cloud or locally without requiring any lower-level operating system support. This approach is justified because most existing applications are designed in this fashion to run either on the cloud or locally, but not both. Furthermore, this work does not make the assumption that both the cloud and local applications are "clones" of each other; in fact, the difference in features between the two versions is considered part of the decision process.

In this dissertation it is assumed that the desirability of any application is a function of the features it offers a user and performance during execution. Features could include application functionality, ubiquitous access, and availability of other users with whom to interact, to name a few. Performance could be measured based on execution speed and response time. For cloud-based applications, network connectivity is an important factor in meeting performance goals. Finally, an algorithm that considers the rating of applications and allows a tunable tradeoff between user experience and energy savings is proposed in this dissertation.

## 3.3 Analytical Characterization

In this section, the formal problem statement and analytical model for application functionality and performance metric called AppScore is defined. Subsequently numerical validation is performed for the model in order to understand the influence of various parameters on user experience of any application. Then an algorithm, GreenSpot, is proposed to manage the energy and user experience tradeoff.

### 3.3.1 Application Score Metric

Consider an application that could be either accessed using the Internet or installed on a mobile device locally, e.g., a Chess game. It is assumed that the application is a function of the features it offers to the end user as well as performance during execution. Let the application performance score (AppScore) be defined as $R$ $(-\infty < R \leq 1)$, a weighted score of the desirability of an application based on the features it offers a user and its performance. This can be expressed as

$$R = w_1 F + (1 - w_1) S \tag{3.1}$$

where $w_1$is a weight used to assign the relative importance between features, with a score of $F$, and performance, with a score of $S$, to a user, and $0 \leq w_1, F \leq 1$.

Performance when running an application on a remote server will be a function of the network conditions, and the capability and load at the server. When running the application locally, performance is a function of its processor capability and load. Thus, the score $S$ $(-\infty < S \leq 1)$ can be expressed as

$$S = \begin{cases} w_2 N + (1 - w_2), & Cloud\ execution \\ \frac{1}{Speedup}, & Local\ execution \end{cases} \tag{3.2}$$

where $N$ $(-\infty < N \leq 1)$ is a score assigned based on network connectivity to the remote server. If network conditions are bad, a negative score can be assigned, thus favoring the use of local applications. The weight used to assign relative importance between network characteristics and processing capabilities when running cloud applications is $w_2$ $(0 \leq w_2 \leq 1)$. This weight could be assigned based on how often the network will be used and processing requirements at the server for a specific application.

The term "speedup," used previously, can be defined as the ratio of server-processing capabilities to local-processing capabilities. With the server capability normalized to unity, the local performance would be $\frac{1}{Speedup}$, with speedup > 0. A speedup value of unity indicates similar processing capabilities locally as well as in the cloud, while a larger value denotes more powerful cloud-based processing. The speedup value can be less than unity when the servers are unable to handle the additional load.

*N* is defined as a function of both packet loss rate *e* and network latency *l* using the relation shown in equation (3.3):

$$N = w_3(1-\frac{e}{t_e}) + (1-w_3)(1-\frac{l}{t_l}) \tag{3.3}$$

where $t_e$ and $t_l$ are application-specific desirable packet loss rate and network latency, respectively, and $w_3$ ($0 \leq w_3 \leq 1$) is a weight used to assign relative importance to one network factor or the other. Applications such as video over user datagram protocol (UDP), for example, are more tolerable to packet loss than latency, and therefore, a higher weight needs to be assigned to the latter. It is assumed that the impact of other network parameters, like available bandwidth, will be manifested as either packet loss or latency; if needed, the model can easily be extended to include other parameters.

### 3.3.2 *Lemma 3.1*

Given the user-preference weight $w_1$ and application-specific weight $w_3$, the feature score $F_c$ for the cloud application and feature score $F_l$ for the local application, and network connectivity score *N*, the minimum speedup required for $R_{cloud} > R_{local}$ is

$$Speedup = \frac{1-w_1}{w_1(F_c-F_l)+(1-w_1)[w_2N+(1-w_2)]} \tag{3.4}$$

***Proof:*** This follows trivially from the expressions of $R_{cloud} = w_1(F_c) + (1 - w_1)[w_2 N + (1 - w_2)]$ and $R_{local} = w_1(F_l) + \frac{1}{Speedup}$ using equation (3.1) and the condition $R_{cloud} > R_{local}$. Note that when the denominator of equation (3.4) is less than or equal to 0, there exists no speedup value where $R_{cloud} > R_{local}$. A condition for $N$ given speedup can be expressed similarly.

The next lemma, Lemma 3.2, provides a lower bound on the difference in features required for cloud-based execution to be more desirable and, if speedup is known beforehand, can prove to be a useful preliminary check before gauging network conditions.

### 3.3.3 *Lemma 3.2*

Given user preference weight $w_1$, feature score $F_c$ for the cloud application and $F_l$ for the local application, and relative processing capability speedup, the minimum difference in feature scores for the cloud application to have a better AppScore than the local application under *any network conditions* is given by

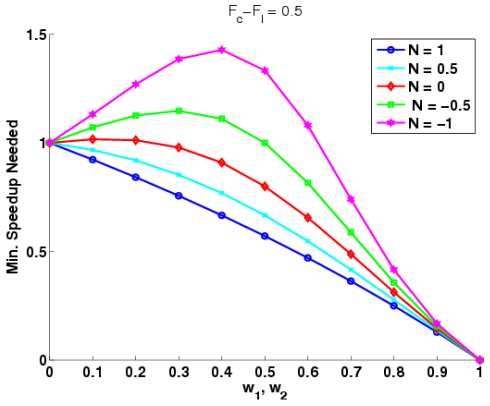
$$F_c - F_l = \frac{1}{Speedup}\left(\frac{w_1}{(1 - w_1)}\right) - 1 \tag{3.5}$$

***Proof:*** This follows from the expressions of $R_{cloud} = w_1(F_c) + (1 - w_1)[w_2 N + (1 - w_2)]$ and $R_{local} = w_1(F_l) + \frac{1}{Speedup}$ using equation (3.1) and the condition $R_{cloud} > R_{local}$. By using the best possible score for network connectivity $N = 1$, the condition reduces to the desired one.

## 3.4 Numerical Evaluation of AppScore

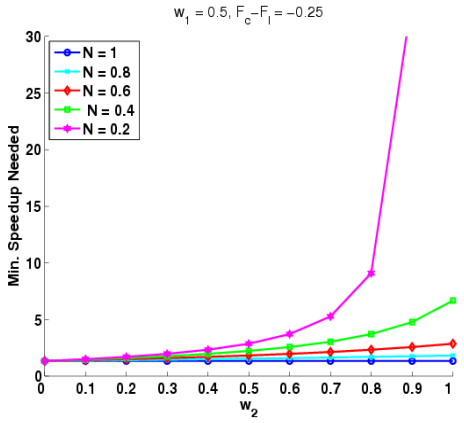
To obtain a sense of the impact of various parameters on AppScores, two sets of numerical evaluations are presented in Figures 3.1 and 3.2. The first set of evaluations mainly shows the minimum speedup necessary for $R_{cloud} > R_{local}$, considering an overall score $N$

without the underlying relative weight $w_3$between packet loss and network latency. The second set looks at all three weights $w_1$, $w_2$, and $w_3$. The plots in Figure 3.1 show examples of how much server speedup is necessary to overcome limitations of the network or application functionality.




(a)

Speedup needed for $R_{cloud} \geq R_{local}$ for varying values of $N$, and $w_1$, $w_2$ ( $w_1$= $w_2$) with $F_c - F_l = 0.5$



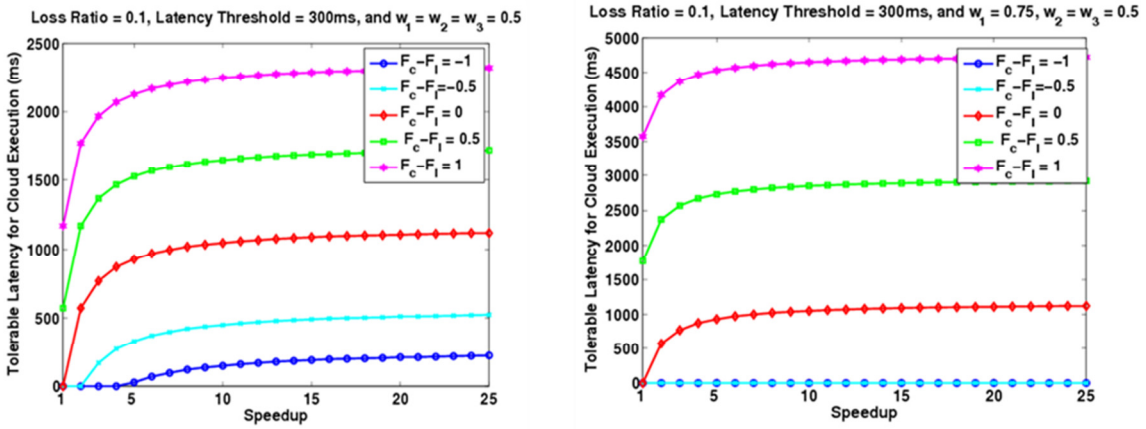

(b)

Speedup needed for $R_{cloud} \geq R_{local}$ for varying values of $N$, and $w_1$, $w_2$ ( $w_1$= 0.5) with $F_c - F_l = -0.25$

Figure. 3.1: Using AppScore to judge whether cloud execution provides necessary speedup over local applications to improve user experience, given network conditions and difference in application functionality.

Figure 3.1(a) shows that when $F_c - F_l = 0.5$, the cloud application score can exceed the local application, even with a network score $N$ as low as –1 for small values of speedup. Figure 3.1(b) shows that when $F_c - F_l = -0.25$, that is, the local application functionality score is higher by 0.25, positive values of $N$ are required. By keeping $w_1$ fixed at 0.5, the plot in Figure 3.1(b) reveals that large values of $w_1$ are the reason that no value of speedup is feasible. When the weight assigned to application functionality is higher, it becomes more difficult for the cloud application to overcome its lack of features, even under good network conditions and high values of speedup.

Because the impact of network connectivity is very critical relative to the decision between a cloud and local application, Figures 3.2(a) and 3.2(b) display the maximum tolerable network latencies for a range of speedup values and differences in features. The ratio of packet loss rate to tolerable loss rate was kept fixed at 0.1 to focus solely on the impact of network latency; the same could be easily done for packet loss. The desirable network latency $t_l$ was kept at 300 ms, which is a roundtrip value. The results for weights $w_1 = w_2 = w_3 = 0.5$, as shown in Figure 3.2(a), show that if the cloud application is lacking in features, it can tolerate very little increase in network latency over the desirable value to provide a good user experience. If features are given more weight by increasing $w_1$ to 0.75, as can be seen in Figure 3.2(b), this effect is more pronounced, with cloud applications *required* to have better features than local applications.




(a) $w_1 = w_2 = w_3 = 0.5$

(b) $w_1 = 0.75, w_2 = w_3 = 0.5$

Figure 3.2: Maximum tolerable network latency for cloud applications to be favorable.

## 3.5 Proposed GreenSpot Algorithm

Based on the empirical analysis and analytical results, presented earlier in this dissertation, an algorithm, GreenSpot, is proposed to help a mobile device determine whether it should run a

cloud or non-cloud version of any application, if the choice is available. The algorithm evaluation is presented here.

The GreenSpot algorithm begins with the user requesting an application. As shown in Figure 3.3, the algorithm checks if both a cloud and a local version of the application are available. If yes, then it proceeds with evaluating the two options. If not, then the algorithm uses the version available. This check is important because users may only have a cloud version of an application (e.g., a device that does not have Microsoft Office with the user relying on Google Docs for word processing).

***GreenSpot* Algorithm**

1. ***If*** only one of the cloud or local version available ***then***
2. Use that version
3. ***end if***
4. ***Get*** weight $w_1$ $F_c$, $F_l$ based on user preferences
5. ***Get*** $w_2$, $w_3$, $t_e$ and $t_l$ based on application nature
6. ***Send*** test packets to cloud server to get *e* and *l*
7. **Compute** *N* and *Speedup*
8. **Compute** $R_{cloud}$ and $R_{local}$
9. ***If*** $R_{cloud} \geq R_{local}$ ***then***
10. **Compute** Power Ratio
11. ***If*** Power Ratio < 1
12. Use cloud version of the application
13. ***end if***
14. ***end if***
15. Use local version of the application

Figure 3.3: GreenSpot algorithm.

The next phase of the algorithm is used to obtain parameter values to compute relative desirability of the application performance and energy for the cloud version and the local version. If the user preference is application quality of experience, then it gathers information on the features available in both versions $F_c$ and $F_l$, and assigns weights $w_1, w_2, w_3$, tolerable loss

rate $t_e$, and latency $t_l$, based on user preferences and nature of the application requested. Test packets are sent to the server to obtain current estimates on packet loss rate and latency to compute *N*, and to fetch information to compute speedup relative to local-device capabilities. If the cloud AppScore $R_{cloud}$ is found to be greater than some factor of the local device AppScore $R_{local}$, then the expected performance to execute the cloud version of the application is deemed satisfactory.

If the user preference is energy, then the algorithm computes the power ratio and uses the condition presented in Chapter 2, equation (2.11), to determine whether the cloud or non-cloud version consumes less energy and, hence, is more preferable.

### 3.6 Greenspot Application Framework and Development Methodology

In order to implement the GreenSpot algorithm, an Android application was developed. This application was installed and tested on an Android cell phone: HTC Desire A8181, version 2.2. Figure 3.4 shows the front end of the main activity page of the GreenSpot application, which was developed for the application in this project.

Based on the GreenSpot algorithm, text views and respective text fields, such as $F_{cloud}$ (information of features available on the cloud version application) and $F_{local}$ (information of features available on the local version application) as well as the application feature weight $w_1$ were created in the front end of the application. Color coding, layout, and style sheet were encoded in the "*activity_main*" XML page. Text views are labels where the name of the text field was entered before the page was loaded. Text fields are text boxes where data was entered after the app was loaded on the Android mobile device. After providing the information in the text fields, the "*ClickHere*!" button, which initiates coding at the backend of the application, was pressed.

In the back end of the application, Android "*Application Packages*" were included in the program for implementing the necessary activities. Some of these packages were Uniform Resource Identifier (URI), Bundle, Intent, Onclicklistener, Button, Edit Text, and Text View. The URI package is mainly used to identify the abstract of any part of a resource. In most cases, when a page is refreshed, all data in the fields will be deleted. The *Bundle* package is generally imported to hold values in the fields for a period of time, even after a page has been refreshed. The *Intent* package is used to start an activity and to mediate between activities.

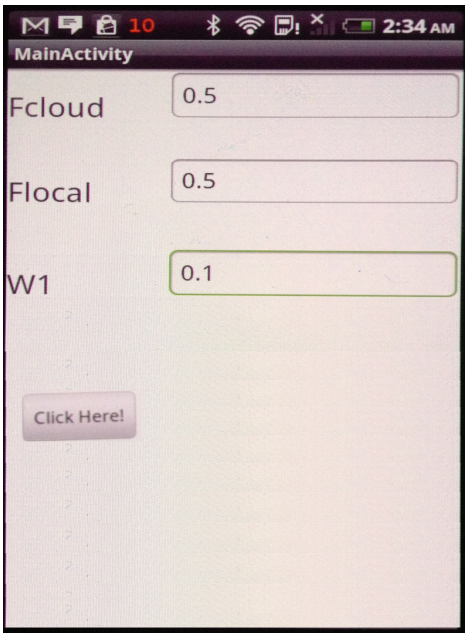



Figure 3.4: Snapshot of main activity page of GreenSpot application.

The GreenSpot application was programmed to redirect the user to the cloud version if the power consumed by the online game was lower compared to the local-version app. The *Intent* package plays a key role during the transfer of activities at the time of redirecting the application. The *View* package is imported for various purposes; it has many interfaces that can be extended based upon the requirements. Other packages, like *Button*, *Text View*, and *Edit Text*, are used as widgets to make controls on an XML page accessible to use that data on the Java page.

The next step in the program was to create a *public* class main activity. All of the graphic layout XML values were initiated in private controls of the Java page, which extends each activity such as $txt\_w_1$ for the textbox of $w_1$ in the XML page. A *main activity* public class was then created, which included the main process of the entire application, and all text controls were appended. Initially the values of the weights, loss rate, and latency values were entered in the "*activity_main*" XML page. In this application, *view* was used to click on events for the methods that must be handled throughout the application. An "*on create*" instance was created to call the function when the "*ClickHere!*" button was triggered. After that, the values were analyzed and calculated in the back end of the program. Using validation controls, these values were verified as to whether the data entered in the textboxes were valid for the application; if the values are not satisfactory, then the application will not allow the user to go through the process until valid data is entered. Once the values were satisfactory, the program underwent the next stage of calculation. The values from the XML edit text were sent to the private edit text values that were initiated in the main activity of the Java page. All entered values were converted to double values because the output obtained was a decimal value. With the values of $w_1$, $w_2$, $F_{cloud}$, and $N$ values, the $R_{cloud}$ value was calculated, and with the values of $w_1$, $w_2$, $w_3$, $t_e$, $t_l$, $e$, $l$, and $F_{local}$ values, the $R_{local}$ value was calculated. All values obtained from the calculations were held in a bundle and forwarded into local double values for other calculations. Then an "*if condition*" was initiated, where $R_{local}$ and $R_{cloud}$ values were compared, based upon the condition (i.e., $R_{local} < R_{cloud}$ or $R_{cloud} < R_{local}$) in the *if loop* it should be directed to the relevant intent. Before placing the if condition, a "*try condition*" was used in this program. This is a function that checks whether the main program contains any lines of code with errors. The main executable program used for this application was placed in the try condition function. Then, in the main if condition, if the $R_{cloud}$

value was greater than the $R_{local}$ value, then new intent was initiated, and the page value was redirected to the $R_{cloud}$ intent page, which was placed in the new intent condition.

In the Greenspot application, two intent conditions were created. The first checks whether $R_{local}$ is greater than $R_{cloud}$, and if the condition is satisfied, then the user gets redirected to the local version of the application; if the condition is not satisfied, then the program enters the second intent condition. The second intent condition was to obtain the power ratio defined in equation (2.11). According to the GreenSpot algorithm, if the calculated power ratio value is greater than 1, then the program is redirected to the cloud version of the application. If the program condition is not satisfied, then the program emerges from the loop and redirects the user to a local version of the application. The main motive of intent was to redirect the program either to the local app or to the cloud app, whichever consumed less power.

A pop-up window coding was placed in the position where it states whether or not the page should be redirected to a cloud app. If the Android user decides not to use the cloud application, even though it is more energy efficient, then the user could send a command to the main program to redirect the program to the local version of the application. Once the user requests to execute the application on the local app, a new window pops up and asks for confirmation of the user decision, as shown in Figure 3.5.

Once the entire coding is completed for *if* and *else if* conditions, then the user must close all necessary syntax for those conditions. After that, a "*catch statement*" must be placed at the end of the application. This catch statement is executed when an error occurs in the main program, and it is placed in the try condition. The main motive of the try condition was to execute the main program in the application, and the main motive of the catch statement was to execute the code that is present when an error is triggered in the main program. In the catch

statement, it is necessary to place a code that redirects the application into a safe zone, or the main page of the application, which contains an output statement saying that an error has occurred in the main program. A "*null pointer exception*" was placed within the catch statement to attempt to identify when the program has obtained any null values during calculations. The probability of obtaining null values during calculations is very low, but certain issues will become risks to the application when null values are obtained during calculations.

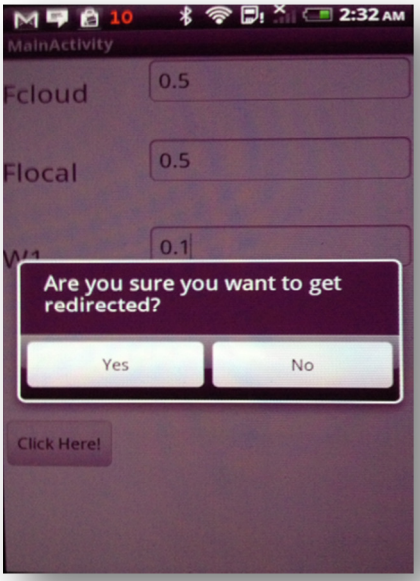



Figure 3.5: Snapshot of user decision confirmation window of GreenSpot application.

For example, if a null value is obtained for $R_{cloud}$ and an effort is made to convert that value into a double value, then this might cause an issue for the entire application because a null value cannot be converted to a double value. To avoid entering null values for the text views, validation controls were used. This validation control prevents users from entering into the main program if null values are entered. However, calculations that might result in null values inside the main program cannot be controlled. Therefore, to avoid these problems, a null pointer exception was used in the catch statement.

The condition "*finally*" executed the entire program, even though there were errors in the main program or there were any null values in the catch statement. This must-execute statement should be placed to close an application or redirect the user to the main page of an application. In the finally statement, intent should be activated, in which an application is closed when it is not redirected to any version of the game app. After the finally statement, all open parentheses for the conditions, statements, and main activity are closed with appropriate messages and information. The overall coding for the Android program was completed by closing all syntax for the entire methods and classes.

## 3.7 Greenspot Application Implementation and Validation

The GreenSpot algorithm was developed as described in section 3.6, which explains the experience of implementing and deploying this app under practical scenarios. An application was created on an Android cell phone, version 2.2, and was tested for two different applications. The application was executed for the steps outlined in the GreenSpot algorithm, which then called as subroutines either a cloud-based application of the game or a local, non-cloud application of the game.

In order to test this product, both the local and cloud applications were executed for the same amount of time, thus allowing the average power usage to dictate the relative energy consumption. The most challenging aspect was getting the relative speedup value without any method implemented at the remote server to provide the information. The best solution without such support would be to run a benchmark that most closely resembles the application workload both locally and on the remote server to compare performance. Without access to such tools at the server, estimates based on prior history of execution speed can be used. The current

implementation uses a simple manual entry of this information based on server response times and knowledge of local hardware.

In general, the parameters needed by the GreenSpot algorithm are device and application specific, or need to be obtained from the network and remote server. The device-specific parameters could be easily obtained at initialization through the operating system or known specifications. The application-specific parameters can be learned through prior history of execution. The first time an application is executed, either as a cloud or a non-cloud version, parameters from this run could be stored along with network conditions at that time. The user could be encouraged to try both versions the first time so that the necessary history can be gathered. Such history for an application could also be shared across users, keeping in mind differences in device type and network conditions.

In order to measure the power consumption on the mobile device, the PowerTutor application was used again. The details of power-measurement methodology are described in section 2.5.1. Power consumption was measured for use in evaluating the power ratio as presented in equation (2.11), for both local and remote program execution, similar to what was presented in section 2.5.2. In order to have a systematic measurement, the user moves of a chess game were recorded and re-executed five times in order to log the power consumed by each case.

The input for packet loss and network latency was determined by sending a series of ping packets at intervals of 20 ms for 2 seconds providing 100 samples to the cloud-based application server. This small delay should be easily tolerable and can be reduced even further with additional testing with lower time intervals between successive test packets or with fewer packets. The feature score was set to 0.5, assuming that both the local and remote versions of the application provide the same feature experience to the end user.

For evaluating the Chess application, GreenSpot used the local application, which was found to save, on average, 42.5% power (with a standard deviation of 7.5%) over the remote version, for a value of $\tau = 1$. On the other hand, for an application that called upon a remote or local app to search a word within a file with the same algorithm parameter settings as the Chess application, it was analyzed that when GreenSpot used the remote application, 5–40% power was saved, with even greater savings for larger file sizes (varied from 30 KB to 90 KB, refer to section 2.5.2) to be searched. These evaluations showed not only the feasibility of implementing GreenSpot practically but also the utility of GreenSpot in reducing energy consumption in mobile devices.

## 3.8 Conclusion and Future Work

To conclude, in this work, an application comparison metric for user experience was developed to evaluate the relative functionality and performance of any applications executed on the cloud or on a local device. Numerical analysis was performed to validate the user-experience metric, *AppScore*, An algorithm, GreenSpot, was developed and deployed on mobile Android device to seamlessly decide between cloud or non-cloud versions of applications. This algorithm considered the AppScore rating of applications if executed over the cloud or locally, and allows a tunable tradeoff between user experience and energy savings. For the case of Chess application, GreenSpot application saved, on average, 42.5% power (with a standard deviation of 7.5%) over the remote version, for a value of $\tau = 1$.

The GreenSpot application was employed to encourage game application users to consider using the less-power-consumption version of the game apps compared to the more-power-consumption version. Thus the above evaluations showed not only the feasibility of

implementing GreenSpot practically but also the utility of GreenSpot in reducing energy consumption in mobile devices.

Future work involving the GreenSpot Android application could be to automatically learn and enter different weights of the algorithm into the program, eliminating this manual entry on the part of the user. The user then would only specify the path of both the cloud and local version applications, automatically capturing the weights based on their version processing capabilities, latency, and accessibility.

# CHAPTER 4

# REDUCING LIFE-CYCLE ENERGY CONSUMPTION FOR MOBILE DEVICES

This chapter begins by introducing the sustainability and globally increasing energy consumption issues due to the growing number of mobile devices. Subsequently related work in the area and a more formal problem statement are discussed. Additional software-based thin-client solutions to the stated problem and empirical evaluation results are presented. In addition to publications under review, three of them related to this work have been published [38, 39, 48].

## 4.1 Introduction

Another aspect of increasing concern in the mobile computing paradigm is the rapid increase in electronic waste generated due to the frequent replacement of such devices. This is driven in part by the doubling of CPU processing power and mass storage in these devices every one and a half years [8]. Due to their shorter life spans of 1–3 years, millions of cell phones and other mobile devices are discarded every year. Only 3–7% of these discarded devices are recycled, while the rest contribute to electronic waste [8, 41]. Decreasing the discard rate of these devices not only reduces waste but also the energy consumed to manufacture new devices to replace existing ones.

Emerging technologies are being designed to try and meet the fundamental human needs while efficiently preserving the global environment. The mobile computing and communication research community is consistently trying to provide more green and sustainable designs for portable devices. Various green approaches like reusing, recycling, and using more recycling material in the devices, virtualization, and thin-client devices etc. have been proposed to reduce e-waste [41]. The aim of a sustainable portable device design is a longer life span, more power efficiency with less toxic hardware, improved user satisfaction, and a simultaneous reduction in

electronic waste. In a report on sustainability by Nokia Inc., sustainability is defined as fulfilling existing needs by attaining proper equilibrium between economic, social, and environmental priorities without compromising future needs [40].

This dissertation proposes that progress on environmental sustainability for the mobile computing paradigm could be achieved by making progress on two fronts: reducing the energy consumed by each individual device (including energy costs to manufacture them) and reducing the rate at which these devices are discarded. One such green, sustainable approach for mobile devices is leveraging the thin-client approach. The thin client (or simply client) is a simplified device with possibly fewer hardware components compared to a traditional mobile device. Its limited capabilities, however, can be greatly enhanced when used with comparatively powerful and capable servers over a network. In the thin-client paradigm, most computation tasks associated with applications are offloaded to a remote server. The client displays the only graphical output. Given the potential benefits of thin clients to move towards sustainable mobile computing, this work tries to answer the following question: *Does using the thin-client paradigm for mobile computing really reduce resource consumption compared to traditional devices?*

## 4.2 Related Work

Most research related to sustainable computing is focused at the data centers. This dissertation work focuses on the sustainability of client mobile devices. The number of mobile devices is increasing globally, which in turn will proportionally increase the total energy consumed by them. An effort is required from the research community to design more sustainable and green mobile devices. According to a report by Somavat et al., information and communication technology (ICT) consumes 3% of the global power usage. Of the total energy consumed by ICT, portable devices are responsible for nearly 17% [1]. Hence, the creation of a

more energy-efficient model for portable devices has been a primary focus for many researchers in the mobile communications sector. The comparatively shorter battery life has also been one of the major concerns of portable devices. Researchers have proposed various technologies like fuel cells, cell-sized batteries, nanotechnology, and Li-ion batteries for improvement of battery performance of portable devices [42]. Consistent increase in functionality of the portable devices also affects the battery energy consumption proportionally, which in turn affects the global energy consumption due to the frequent charging of batteries.

Mobile communication technology is also partly responsible for the emission of gases like $CO_2$, one of the principal causes of global warming. A single smartphone could be responsible for 9 Kg of $CO_2$ emissions, including three years of product usage phase. The impact of greenhouse gases due to a single smartphone is equivalent to the impact of driving a car for 54 km [41]. It was observed in the life-cycle assessment of these devices that most $CO_2$ emissions come from the production and usage stages. During the usage stage, the base station was reported to emit most of the $CO_2$ [43]. Somavat et al. [1] reported that the computing sector is responsible for 270 million tons of $CO_2$ emissions every year. This is due to the fact that all computing sector devices operate on electricity, which is primarily produced using coal. Such challenging environmental threats from mobile communication systems and a formidable growth in user demand for portable devices motivate researchers to look into sustainability in mobile computing.

In view of developing a reliably green and sustainable solution, the work in this dissertation proposes a model for reutilization of old mobile devices with the help of a cloud-based thin-client approach. The thin-client paradigm primarily operates on the concept of

offloading a majority of computations to a remote server for execution of the application logic. The client, in most cases, is used for displaying the graphical output only.

Mobile thin clients are simplified devices with considerably less hardware components than traditional mobile devices. The thin-client-server model primarily focuses on the enhancement capability of the mobile devices by providing full functionality to the device through a connection to remote servers. These devices could be compared to the dumb terminals of the mainframe era in terms of limited features and hardware capabilities on both machines. Thin clients provide several benefits over traditional mobile devices through access transparency based on location due to mobility, increase in personal data storage, resource-intensive applications executed on the cloud, and ease of maintenance and upgrading [45, 46]. The thin-client-server approach is a technology that is being adapted by industry to reduce manufacturing and maintenance costs of devices while effectively trying to meet user requirements in terms of performance and reliability. Thin clients are becoming prevalent due to low cost of the required hardware and increased lifespan of the devices. A noteworthy feature of the thin-client server model is the capability to decisively perform an application execution either locally on a mobile device or remotely on a server based on the availability of resources and complexity of the required computation [47]. Thus the thin-client server model provides a major advantage by promoting energy-efficient usage on the local mobile device by diverting the computation intensive tasks to a remote location.

Although thin clients have been broadly evaluated in the literature for their performance and the user experience, rarely have they been analytically studied via a mobile computing paradigm approach, which is not only green in terms of reducing e-waste but also economical in terms of energy consumption. This work presents a semi-empirical analytical model to evaluate

performance improvement, end-user quality of experience, and energy savings, which could be achieved by reusing old or discarded mobile phones under a thin-client paradigm. The model presented here is based on various critical parameters such as life-cycle energy, energy consumption on various levels of the life cycle, performance improvement of old mobile devices, etc. The novelty of this work is accentuated via the life-cycle energy model for mobile devices, which would provide a comprehensive insight into where and how energy savings could be achieved in the overall life cycle of the device. Additionally, the life-cycle model fine tunes a path for steering the design toward a fully sustainable structure.

Subsequently the analysis of energy savings and user experience under a thin-client paradigm for reusing mobile devices is also modeled and evaluated. This cloud-based approach focuses on the reduction in manufacturing and maintenance energy costs of thin-client mobile devices. An analysis of the performance and speed-up improvement for thin-client mobile devices is also presented in later sections.

Specific contributions of the work outlined in this chapter include the following**:**

1. An analytical model for life-cycle energy consumption of mobile devices was proposed to evaluate the energy consumption during various phases of its life cycle.
2. Numerical analysis was performed to validate the proposed analytical model and to identify the impact of increase in number of usage stages on the life-cycle energy cost of a device, which could be achieved by reusing old smart phones under the thin-client paradigm.
3. An analytical characterization was performed to identify the theoretical conditions for energy savings under the thin-client paradigm.

4. An empirical comparison for performance and energy consumption for application execution under the thin-client paradigm was performed to validate the proposed analytical model using Android smartphones.
5. An Android application was developed and implemented on various Android smartphone devices. Energy savings, performance, and speedup analysis for reusing old smart mobile devices obtained via the experimental tests under thin-client execution of an application on smartphones is presented in the later sections.

## 4.3 Mobile Device Life-Cycle Energy Consumption

The following section discusses the analytical characterization via the proposed life-cycle energy consumption model. The model was designed to be generic enough so that it could be implemented on any mobile device. The goal of the proposed analytical model is to identify the impact of increase in number of usage stages on the overall life-cycle energy cost of a mobile device. In the subsequent section this model will be used to evaluate the performance and energy consumption of devices such as smartphones when reused under a thin-client paradigm.

### 4.3.1 Proposed Model for Life-Cycle Energy-Consumption Analysis

In this model, it was assumed that the total energy consumed throughout the life cycle of a mobile device consists of three major components: manufacturing, usage, and recycling energy. Consistent with these assumptions, three major life-cycle phases along with the corresponding phase energies defined as manufacturing energy ($e_m$), energy consumed during the usage phase ($e_i$), and recycling energy ($e_r$) were considered in the theoretical model. Additionally, it was assumed that mobile devices could be reused. Moreover, the reutilization of these devices could consist of several usage stages ranging from 1 to $n$, as shown in Figure 4.1. Parameters defining the model are shown in Table 4.1.

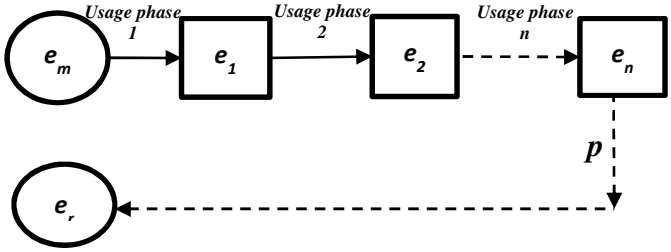



Figure 4.1: Model considered for sustainable mobile devices.

TABLE 4.1

DEFINITIONS OF PARAMETERS USED IN MODEL

| Variable | Parameter Definition |
|---|---|
| $e_{Total}$ | Total life-cycle energy of mobile device |
| $e_m$ | Manufacturing energy of mobile device |
| $e_u$ | Usage phase energy |
| $e_r$ | Recycling energy |
| $p_i$ | Probability of exiting to recycling phase |
| $z_i$ | Fraction of manufacturing energy required for replacement of components |

Each device begins with a manufacturing phase associated with a manufacturing energy ($e_m$). The device then traverses through various usage stages in the life cycle based on the probability to exit to the recycling phase after completion of each usage phase. It was assumed that each device mandatorily enters the first usage phase. The device then traverses through various usage stages in the life cycle based on the probability ($p$) to exit to the recycling phase after completion of each usage phase, as shown in Figure 4.1. The recycling phase is associated with a recycling energy ($e_r$) which also contributes to total life-cycle energy consumption.

Considering that the probability to exit to the recycling phase is $p$ and $n$ is the number of independent usage phases which the device traversed before exiting into the recycling phase at

the $n^{th}$ usage phase. Thus the probability mass function of any device exiting to the recycling phase at the $n^{th}$ usage phase is defined as

$$pmf = (1-p)^n p \quad \{n=1, 2, 3\ldots\ldots\}, (n>0) \tag{4.1}$$

Considering a geometric probability distribution, the total number of times the device was reused, including the first usage stage, was calculated by deriving the expectation value $E(X_n)$ as defined in equation (4.2), where $X$ is a random variable between 1 and $n$:

$$E(X_n) = \sum_{n=1}^{\infty}(1-p)^n p.n \tag{4.2}$$

A notable challenge in reusing smart mobile devices is some of the wearing out of hardware components over long-term usage of the device. Typical problems that may be encountered due to long-term usage of these devices are damage to the touch screen display, malfunctioning or damage of the touch screen sensors, end of battery lifetime, etc. Thus, it is assumed that in order to reuse the smart mobile devices and consequently execute any task under the thin-client paradigm, the device must have the following hardware components in a functioning state: LCD touch screen, WiFi module, mother board, and battery. However, other hardware components such as cameras, GPS, and audio are considered task-dependent on the device being reused.

Considering the degradation of hardware components due to reusing the device, a hardware upgradation energy cost ($z_i e_m$) required to replace any components, was also included in the proposed model. Here, $z_i$ is the fraction of manufacturing energy required for replacement of hardware components and varies between 0 and 1. Thus, the life-cycle energy consumption at each usage phase ($e_i$) is the sum of energy consumed at the *ith* usage stage($eu_i$) and the hardware upgrade energy cost ($z_{i-1} * e_m$) involved at that usage stage. The energy consumption at the *ith* usage phase is defined as

$$e_i = eu_i + z_{i-1} * e_m \qquad \{i=1, 2, 3\ldots\ldots\ldots n\}, z_0 = 0 \tag{4.3}$$

Therefore, the total energy consumed by the mobile device throughout its life cycle includes the manufacturing energy cost, energy cost of each usage phase, hardware upgrade cost involved each time the device is reused, and recycling energy cost, as shown in equation (4.4):

$$e_{Total} = e_m + \sum_{i=1}^{\text{ciel}[\text{E}(\text{X}_\text{n})]}(e_i) + e_r \tag{4.4}$$

The average life-cycle energy of the device was calculated using equation (4.5) for the derived expected value of usage stages using equation (4.3):

$$LCE_{avg} = \frac{1}{\text{ciel}[\text{E}(\text{X}_\text{n})]}\,[e_{Total}] \tag{4.5}$$

Thus, the total energy consumed throughout the life cycle of the mobile device can be stated as in equations (4.1) to (4.4).

### 4.3.2 Numerical Analysis

A numerical analysis based on the presented model was performed for smartphones using equations (4.3) and (4.5). The data for analysis was taken from Nokia [41]. The total life-cycle energy of smartphones was evaluated for various selected geometric distributions of $p$. The geometric distribution probability model allows one to determine the number of usage stages a device traverses before exiting to the recycling phase. In order to evaluate the average number of usage stages, an expectation value $E(X_n)$ was calculated for each geometric distribution, where $X$ is a random variable between 1 and $n$. Figure 4.2 depicts the variation of $E(X)$ with the average life-cycle energy. The total life-cycle energy of the device includes manufacturing energy, total usage energy cost (includes multiple usage and hardware upgrade cost), and recycling energy cost, as shown in equation (4.4).

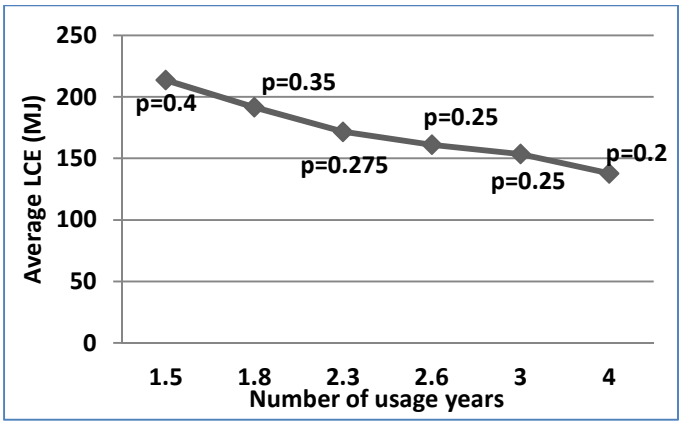



Figure 4.2: Average LCE vs number of usage years.

The average life-cycle energy of the device was calculated using equation (4,5) for the derived expected value of usage stages. It was assumed that each usage stage is 12 months long. It is estimated that the commercial cost to replace components such as the LCD touch screen or battery is nearly equal to 5–20% of the cost of the new device. Therefore, the numerical evaluation includes a hardware-upgrade energy cost after each usage phase, as shown in equation (4.3). An upgrade cost of 5% was added at the end of each usage stage in those cases when the device enters another usage stage considering the hardware degradation due to multiple usages of the device. Figure 4.2 shows the average life cycle energy consumption for various ($p$) probabilities to exit to the recycling phase. It can be observed from Figure 4.2 that the average life-cycle energy consumption decreases linearly as the usage phase increases from 1.5 years to nearly four years. Also devices with lower probability ($p$) to exit to the recycling phase undergoes more number of usage phases. Approximately 11% reduction in the average LCE was observed as a result of three months increase in the usage phase from 1.5 to 1.8 years. This is due to the fact that with less usage stage duration, there exists a disproportion between the manufacturing energy and usage stage energy consumption. Thus, more sustainable devices can help achieve proper balance between the manufacturing energy and usage stage energy.

Next, the thin-client paradigm for reusing mobile devices is a cloud-based software approach in order to increase the number of usage phases by providing the required upgrade through the cloud. Also, this work investigated if older smartphones could provide a comparatively similar performance when a task is executed on both new devices and older devices under the thin-client paradigm.

## 4.4 Thin-Client Approach for Mobile Devices and Experimental Methodology

Recalling equation (4.5), it can be noted that the average LCE is directly proportional to the total energy consumed throughout the device's life cycle. It was also observed from the analysis in section 4.3 that increasing the number of usage stages and reducing the usage-phase energy would effectively result in a reduction in the average LCE. The evaluation of performance and energy savings during reusing the device is crucial in order to comprehend the effects of system parameters such as processing capability and quality of experience. A thin-client server model for mobile devices is primarily based on operating in conjunction with a remote server for resource sharing, accessibility of features, and functionality [48]. This section presents a case for evaluating performance improvement and energy savings when mobile devices are reused under a thin-client paradigm. This dissertation also defines the performance improvement of old mobile devices in terms of speed up and end-to-end computation time to execute any task under a thin-client paradigm.

### 4.4.1 Thin-client Approach for Mobile devices

Thin-clients are simplified devices with conceivably fewer hardware components compared to traditional mobile devices. Capabilities of mobile devices could be greatly enhanced when used with relatively high performance server clouds over a network. In the thin-client paradigm most of the computation tasks associated with applications can be offloaded to a

remote cloud server as shown in Figure 4.3. The thin-client or (simply client) displays only graphical output. The offloading approach also improves the user experience by increasing the execution speed of the tasks via more capable remote servers. *ChromeBook* by Google are a step in this direction where there is only a limited operating system locally while most of the functionality is provided by remote cloud servers.

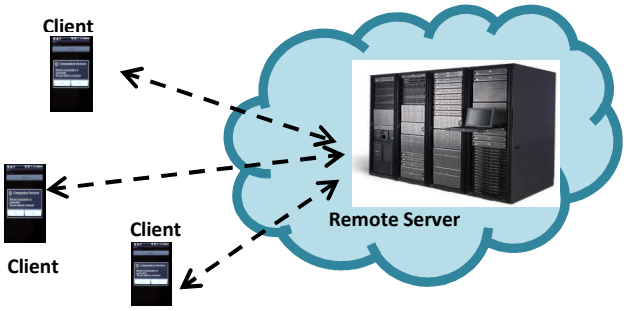



Figure 4.3: Thin client approach for mobile devices

Thin-client server paradigm for smartphones can be effectively implemented for reutilization of old smartphones thereby reducing the overall energy consumption and e-waste due to these devices. In this approach the smartphones must have the active network interface to get connected to the remote server. In order to execute any application in thin- client server approach the client device is required to set up a session with the remote server using TCP/IP socket interface.

The socket interface attaches the client application processes to the remote server. Data is sent and received by the client to the remote server through the socket. To implement the client server TCP/IP socket at application level for thin client approach first the socket is created on the client. Next a transport address or port number is assigned to the socket. Then the client device sets up connection with the remote server on the specified port number. Before the client connects the server, the remote server must have a socket ready to accept the client connection.

Once the server accepts the socket the client and server can communicate to send and receive any data over the network. The next section explains the implementation of mobile thin client approach for reutilization of old smartphones.

### 4.4.2 Experimental Methodology

The proposed model was experimentally implemented by means of a laboratory-based setup of the thin-client scenario. Assortments of tests were performed in order to evaluate the accuracy and applicability of the proposed model. The goal of this experimental study was to compare and analyze the performance of reused mobile devices operating under a thin-client paradigm with that of newer mobile devices having superior hardware capabilities. In this work, an empirical analysis on multiple smartphone mobile devices was performed to evaluate the performance and energy savings for mobile devices executing applications under a thin-client paradigm. An evaluation of the traditional local execution of the application on the mobile device was also developed, as will be presented later in this section.

A well-known limitation for testing the proposed thin-client server architecture for the prevailing applications is the lack of open-source applications. In order to overcome such challenges, an Android-based application similar to that of a typical word-search application was developed for this dissertation. This application was used to perform word searches for a preselected word or phrase in the input word file. The application was also equipped with a counter that keeps track of the number of appearances of the selected word or phrase in the entire input file. The application uses a regular expression-matching pattern to locate the searched set of words in the input file. Input files consisting of different sample sizes (total number of words) were considered for improved accuracy of the test results. The test-word files were stored in an asset folder separately on the local device as well as on the remote server for access during

application logic executions. *Eclipse IDE* was used for Android programming to facilitate automatic rebuilding of the project each time changes were made to the source code [28]. The Android Development Tool plug-in was installed on the Android 2.2 platform for implementation of the algorithm and development of the test applications. The *apk* file was installed on multiple-client devices to execute and test the applications [26]. The client mobile devices used for the empirical testing are listed in Table 4.2. The HTC Desire, HTC One V, and Motorola MB865, with approximate usage ages of 3.5, 1, and 1.5 years, respectively, were chosen as the test devices. Motorola's MB865 is the higher-end device, with nearly double the size of random access memory (RAM) and about 0.2 GHz more CPU clock frequency as compared to the other two devices.

TABLE 4.2

SMARTPHONES USED FOR EMPIRICAL STUDY

| Model | Device Age (years) | Android Version | Specifications | |
|---|---|---|---|---|
| | | | CPU Frequency | Size of RAM |
| Motorola **MB865** | 1.5 | 4.0.4 | Dual-Core 1.2 GHz Cortex-A9 | 1 GB |
| **HTC One V** | 1 | 4.0.3 | 1 GHz | 512 MB |
| **HTC Desire** | 3.5 | 2.2 | 1 GHz Scorpion | 576 MB |

## 4.5 Framework for Benchmark Application Execution

In order to provide a methodical comparison of power consumption for local computation-based and thin-client paradigm-based executions, the same applications were implemented in each test measurement after installing them on the local device as well as at a remote location on the server. The experiments consisted of search operations performed to find the word "Android" in the text files with varying sample size. Table 4.3 presents the test cases, text file sizes, and number of appearances of the preselected word/phrase in the text files for each

case. During the experiments, all sync services on the device were disabled in order to isolate the current operation. The Wi-Fi and 3G interfaces were turned off while executing the applications locally. The Wi-Fi interface was the only external access application that was enabled to run while executing the search operation remotely.

TABLE 4.3

TEST CASES EVALUATED FOR EMPIRICAL STUDY

| Test Case | File Size (kB) | Number of Appearances of Preselected Word/Phrase |
|---|---|---|
| 1 | 10 | 56 |
| 2 | 20 | 112 |
| 3 | 25 | 140 |
| 4 | 75 | 420 |
| 5 | 125 | 700 |
| 6 | 250 | 1400 |
| 7 | 650 | 3500 |
| 8 | 1016 | 5822 |

### 4.5.1 Framework for Local Application Execution on Device

The application logic for traditional or local execution of the applications was written for implementation on the client device, i.e., the smartphone. As a typical characteristic of all smartphones, localized programs utilize the CPU and other resources of the device for computations. A simple graphical user interface consisting of a button labeled “Local” was developed via Android programming, as shown in Figure 4.4(a). Each click of this button generated an event followed by the extraction of the *input file* in the asset folder. The name of the file in which the word/phrase to be searched was entered via the GUI. The word-searching

operation was performed on the entire document using a regular expression, and the count was maintained using a *for loop* that traverses the *input file* line by line.

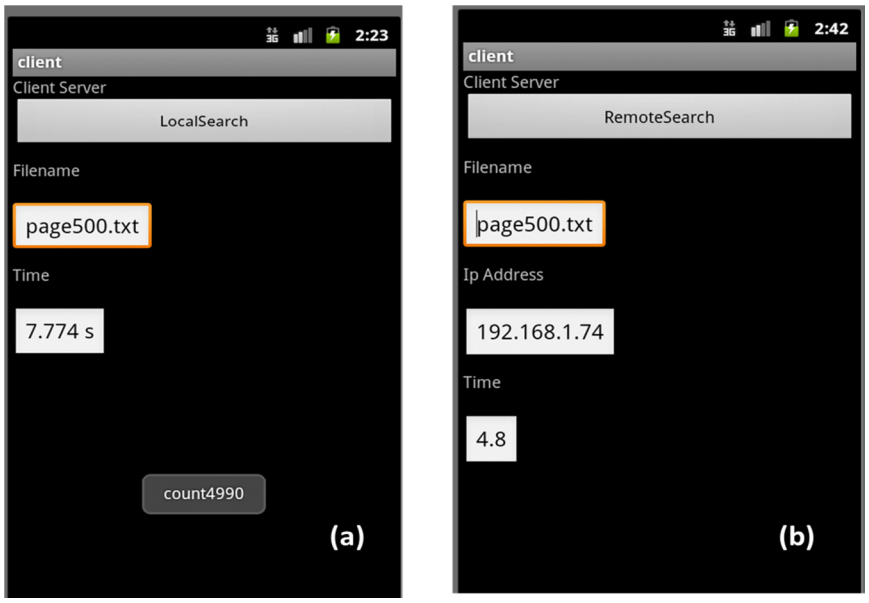



Figure 4.4: Word search benchmark application GUI: (a) local and (b) remote.

### 4.5.2 Framework for Remote Execution of Application via Thin-Client Scenario

In the thin-client application framework, the program is initiated on the client device, i.e., the smartphone, but the computation is executed on the remote server. The remote execution *client and server* scenarios were implemented using socket programming, as shown by the design in Figure 4.5. The Android smartphone was installed to act as the client. In this experiment, the remote server was a laptop equipped with a Core i5 @ 2.3 GHz 64-bit processor, 4 GB RAM, 500 GB hard disk drive, and the client was an Android smartphone with different specifications as shown previously in Table 4.2.

Similar to the setup for the local execution scenario, a simple GUI was developed. This consisted of a button labeled *Remote*, as shown previously in Figure 4.4(b). The GUI included a field to enter the name of the file under testing and the IP address of the remote server.

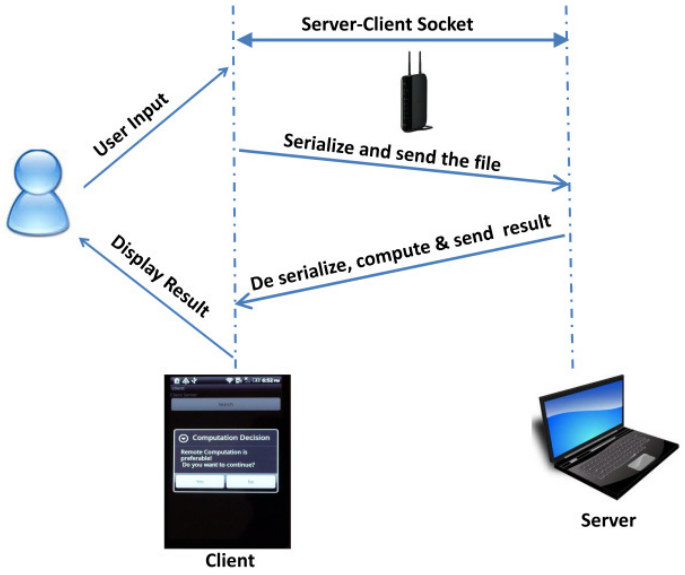



Figure 4.5: Thin-client paradigm for executing applications.

With the *On Click* button, an event is generated in which the input file in the asset folder is extracted to retrieve the values to be computed. The input file extracted from the asset folder is then sent as a byte stream to the remote server as *ObjectInputStream*. After the client-requested task is completed, the sever sends the results back to the client. The final result is displayed on the client device (smartphone). In order to reduce the network delays, the server and the client were designed to reside in the same IP subnet. All experiments were performed under the same network conditions.

## 4.6 Evaluation of Performance Improvement

This section presents the analysis of performance improvement in terms of speedup, as defined in section 4.3, for reusing the mobile device under a thin-client paradigm. First, the speedup measurement methodology is presented, followed by results and analysis.

### 4.6.1 Measurement of Speedup in Local vs Thin-Client Execution

In order to measure the end-to-end time expended by the test application, Android's *Traceview* tool was used. This tool generates trace files with a graphical presentation along with

the detailed CPU time lapse analysis after execution of the application. In the time panel of the tool, each method from the program is represented by a different color code. The method executions in the program are shown in chronological order in a timely manner, with time represented in milliseconds. The profile panel in the tool displays all the synopses of time being spent in the method. It also displays the inclusive time (time spent by the parent method and the time spent by the child method in milliseconds) and the exclusive time (time spent by the method alone in milliseconds). Because the inclusive CPU time comprises the time spent in the parent method and the time spent in any child method (called function), the inclusive CPU time was the time factor parameter considered for this experiment. To measure the CPU time consumption, tracing was performed from the time instance of input data file extraction until the result was extracted back by the application program.

In the case of a local computational program, the time was logged from the click of the "Local" button (event generation for computation) until the result was generated. For the remote computational program, the log included the time track from the time instance of the click of the "Remote" button (event generation for computation), the time taken for sending the file as an object to the remote server (laptop) until the result was generated and sent back to the calling program on the client (smartphone). This remote execution time also includes the time to send and receive the file to the remote server, the time required by the remote server to process the task, and the network delays present in the network. After the completion of the application execution, the trace log file ".trace" was generated and saved with the specified name in the secure digital (SD) storage card of the Android phone.

### 4.6.2 Performance Comparison of Local and Remote Execution Scenarios

The plots in Figures 4.6 and 4.7 show the computation time for execution of the word-search operation versus file size for local-based and thin-client-based scenarios, respectively. The time required for remote executions is the total time ($t^R$), which includes the time for remote processing, sending, and receiving the final result. It can be clearly seen in Figure 4.6 that in the case of local executions (on the smartphone) of the application, the total CPU time for computations increases exponentially with increasing file size.

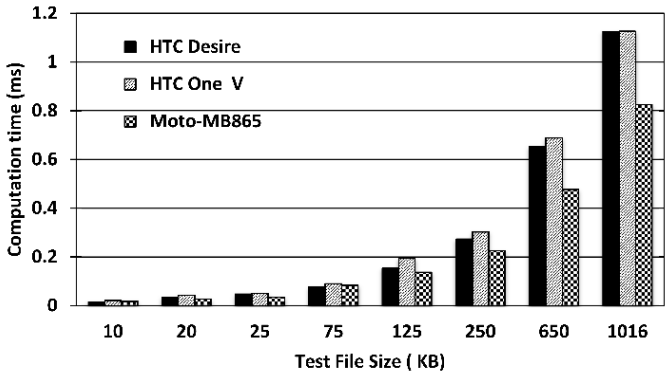



Figure 4.6: Relative performance of traditional local execution (time and file size).

As shown in Figure 4.6, the computation time required for the relatively older devices, such as HTC One V and HTC Desire, increased by approximately 11 times as the file size grew from 10 kB to 1016 kB. However, for the same variation in file size, the computation time for the Motorola MB865 increased by only approximately 8 times, pertaining to its higher-end CPU and RAM capabilities. A significant result is the nearly 30% lower computational time required by the comparatively new Motorola MB865 to process the larger files in the range of 650 kB and ~1 MB, as compared to the other two devices with higher-usage lifetimes. Hence, the Motorola MB865 relatively dominated in performance for the case of local execution of the application as a result of its superior hardware capabilities.

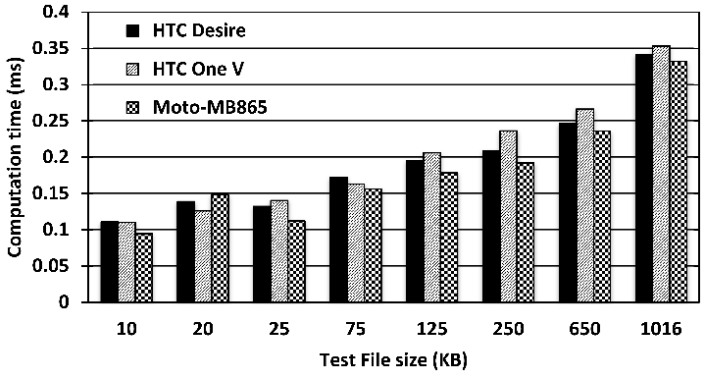



Figure 4.7: Relative performance of thin-client remote execution (time and file size).

Under the thin-client paradigm (remote execution), CPU utilization time increased almost linearly with the increase in file size, as depicted previously in Figure 4.7. The maximum computation time in each test case for all devices was significantly lower in the case of remote execution, irrespective of the file size. Figures 4.8(a–c) depict the magnitude of reduction in computation time via remote processing of the application for HTC Desire, HTC One V, and Motorola MB865, respectively. Such a low CPU time utilization in the thin-client scenario is a direct consequence of comparatively fewer number of computations executed by the local smartphone. The CPU, in this case, is only utilized for sending input data to the remote server and for subsequently displaying the final results.

Another notable observation from previously shown Figures 4.6 and 4.7 is that even though the newer Motorola MB865 possesses superior processing capabilities, it requires about 0.82 ms for processing the ~1 MB file. However, the computation time required to process the same file via the comparatively older smart phones (HTC Desire and HTC One V) through remote executions is 0.32 ms and 0.34 ms, respectively. This computation time for remote execution via the smartphone with higher usage lifetimes is lower by nearly 60%. Such a beneficially high reduction in computation time shows that those devices with a higher usage lifetime could result in efficient performance with the help of a thin-client server paradigm.

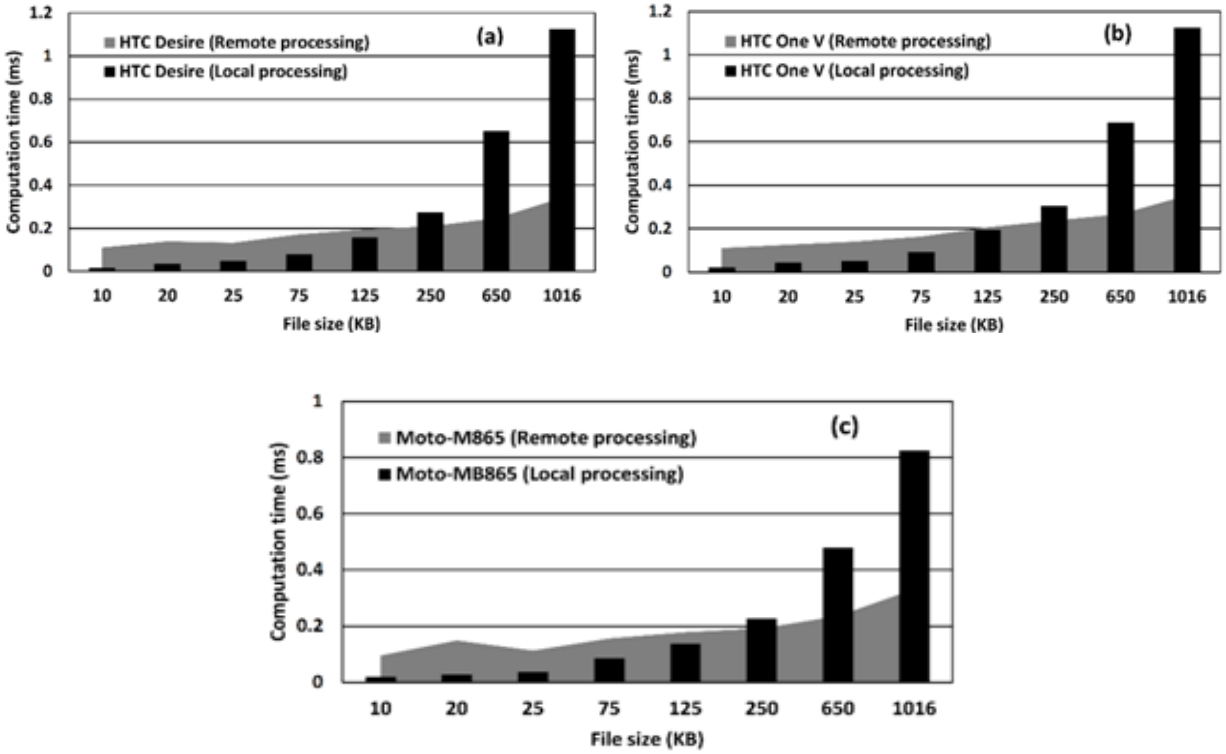



Figure 4.8: Computation times versus file size for remote and local execution scenarios: (a) HTC Desire, (b) HTC One V, and (c) Motorola MB865.

Such benefits essentially reduce the probability of discarding used smartphones and in turn help in lowering e-waste by enhancing the reutilization capabilities of the devices. Results also indicate that the reutilization of these devices would not significantly affect the end-user experience because most of the resources in the local device will only be used to display the results of the remote computations.

The beneficial reduction in processing time via the two older devices (HTC One V and HTC Desire) while remotely processing files under the thin-client paradigm was found to be particularly significant for larger file sizes, as depicted in Figure 4.9. As shown comprehensively, for processing of larger files, the reutilization of the chosen older devices

under the thin-client paradigm results in lower computation times, compared to performing the same tasks locally on newer devices equipped with better hardware capabilities.

As can be seen in Figure 4.9, over a file-size range of 10–75 kB, the computation time required for local processing on the Motorola MB865 is nearly 90% lower than that of remote processing via the HTC smart devices. However, the computation time required by the Motorola MB865 for processing larger files significantly escalates. Conversely, the older HTC smart devices—One V and Desire—require a remarkable 50% and 57% less computation time, respectively, to process the 650 kB and ~1 MB files via remote processing compared to local processing of the same files through the relatively newer Motorola MB865. The notable escalation in computation time above a file size of 250 kB arises from the fact that the number of words to be searched in the larger file size also increases proportionally (see previous Table 4.3), which in turn increases the computation time and load on the processor. Moreover, it should also be noted that in the case of remote processing, the end-to-end time is dependent on the network latency and the processing capability of the remote server.

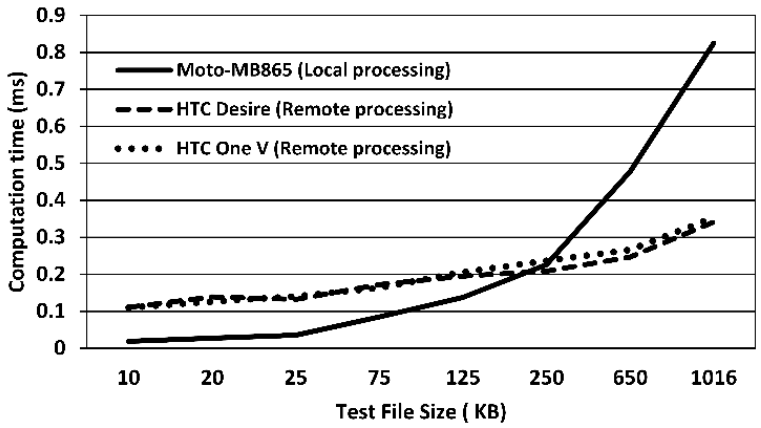



Figure 4.9: Comparison of CPU computation time versus file size for remote processing on HTC (higher usage lifetime) and Motorola MB865 (lower usage lifetime) devices.

Furthermore, efficiency of the thin-client paradigm was analyzed by calculating the speedup defined in equation (2.10), based on the experimentally obtained computation times. Speedup effectively provides an estimate of the relative improvement in computation time under the thin-client paradigm. Figure 4.10 comprehensively depicts the calculated speedup versus file size for each device tested. It is expected from the definition of speedup in equation (2.10) that the lower the time for remote processing ($t_R$), the higher the speedup. All three devices tested showed increasing speedup with an increase in file size. However, it can be seen that for file sizes equal to 125 kB or higher, the speedup for the two HTC devices (higher usage lifetime) is consistently higher than that of the Motorola MB865 (low usage lifetime). The dominance in speedup of the HTC Desire was particularly higher by 25% and 65% for file sizes of 650 kB and ~1 MB, respectively.

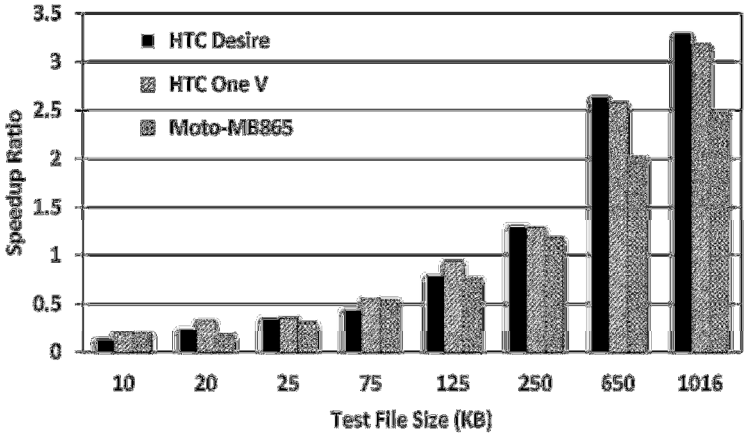



Figure 4.10: Calculated speedup under thin-client paradigm (remote processing).

The inferior speedup for higher file sizes in the case of the Motorola MB865 is a direct result of the low ($t_R$~$t_L$) compared to the HTC Desire, as shown earlier in Figures 4.8(a) and (b), respectively. It should also be noted that according to equation (2.8), $t_R$ consists of both the time required to transfer the file for remote processing (network latency) and OTT time, as discussed

earlier in section 4.2. Hence, $t_R$ varies for each device tested, even though the remote server hardware specifications remain unchanged.

## 4.7 Estimation of Energy Savings under Thin-Client Paradigm

Any increase in the number of usage years results in the manufacturing cost being evenly distributed among the total number of years of usage. As a result, the average life-cycle energy for any mobile communication device could therefore be lowered by the following:

- Reducing the manufacturing energy of these devices.
- Lowering the energy consumed during the use phase.

### 4.7.1 Reduction in Average LCE by Reducing Manufacturing Energy

Thin-clients are a well-known technique for reducing the hardware utilization in desktop-level configurations. However, the same technique has not been widely explored for mobile devices. Thin-client mobile devices use cloud-based resources to enhance their functionality. Such an approach for mobile computing devices could also facilitate the increase in storage, memory, processing power, and battery life of these devices. Thus, these devices would eventually require less hardware and in turn directly cutting the costs for manufacturing hardware components. Such a reduction in hardware components will result in lowering the bill for hardware component materials.

Analytically, the total energy required to manufacture the hardware is directly proportional to the number of hardware components ($n$), as described by equation (4.6). It is assumed that the energy cost for each component is equal to ($E_{C_k}$). Therefore, the manufacturing energy is the summation of the cost required for all the components as defined by

$$E_m = \sum_{k=1}^{n} E_{C_k} \tag{4.6}$$

Typically, the number of hardware components required for thin-clients ($p$) is less than the hardware components in fully functional mobile devices ($q$) i.e. $p < q$. The energy costs for manufacturing each hardware component for a fully functional mobile device and for a thin-client mobile device are represented as $E_{c_k}^{\ f}$ and $E_{c_k}^{\ t}$, respectively. Thus, the theoretical limit for manufacturing energy savings costs is defined as

$$E_{m_{saved}} = \sum_{k=1}^{p} E_{c_k}^{\ f} - \sum_{k=1}^{q} E_{c_k}^{\ t}, (p < q) \tag{4.7}$$

A numerical analysis was performed via equation (4.7) for smartphones to evaluate the predicted manufacturing energy reduction for thin-client devices. In order to calculate the manufacturing energy savings, the data was taken from Nokia [41] and Gobry [4]. The predicted savings in manufacturing energy for a 1–2% reduction in hardware is shown in Figure 4.11.

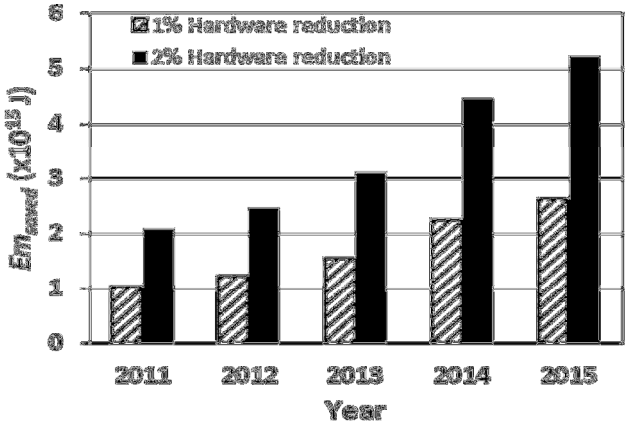



Figure 4.11: Predicted manufacturing energy savings for thin-client mobile devices under various reduction rates in hardware components.

It can be seen that mere 1–2% reductions in hardware components could possibly save energy in the magnitude of millions of Joules per year. Assuming a compound rate of 30% increment in the sales of smartphones every year, the thin-client approach for smart phones could possibly benefit in saving significant levels of manufacturing energy for these devices.

### 4.7.2 Evalutation of Energy Savings

Based on the experimentally obtained $t_R$ and $t_L$ for all devices tested, the percentage energy savings was calculated using equation (2.12). The corresponding power consumption in each case was measured using the Monsoon power meter and a commercially available power measurement application called PowerTutor [23, 24, 25]. A subtractive method was used to log power using the Monsoon power meter. Initially all sync services, WiFi and 3G interface were turned off, and the base power consumption was logged. Power was logged for each individual test case for both local and remote execution of the applications on multiple devices.

In order to calculate the total power in the local ($P_{tot}^{L}$) and remote ($P_{tot}^{R}$) execution scenarios, similar techniques were used as described in section 2.5.1. Figure 4.12 shows the calculated percentage of energy savings versus file size for the test-bed setup under the thin-client paradigm. It can be seen that for all devices, the percentage of energy savings is negative for smaller files, thus indicating that no energy could be saved for these test cases. It was analyzed from the power logs that in this case, more energy was consumed by the wireless interface of the device while communicating with the remote server. However, in the test setup, energy savings retained a positive value for file sizes above 75 kB. This indicates that energy could only be saved on mobile devices under the thin-client scenario if the processing energy consumed on the local device was greater than the energy consumed by the wireless interface to process and send data for remote processing. It was also observed that for the ~1 MB file size test case, approximately 85% of the energy could be saved under the thin-client-server paradigm for all devices, as shown in Figure 4.12. Such a high percentage of energy savings was achieved by offloading a major fraction of the computations to the remote server. However, as per the test setup, there exists a

trade-off between wireless communication for remote processing and local computations for smaller file sizes.

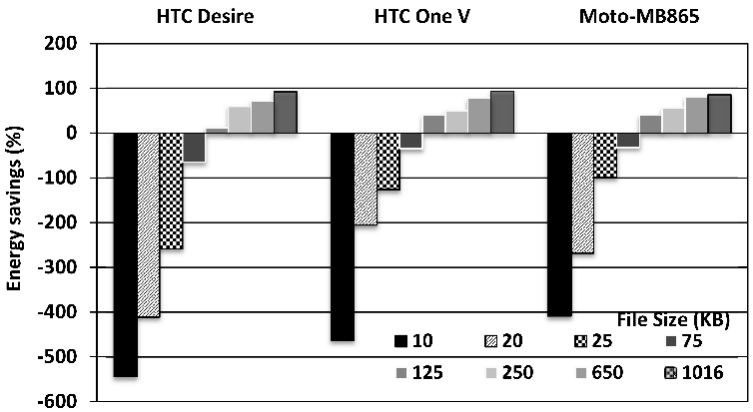



Figure 4.12: Percentage of energy savings versus input file sizes for devices tested.

This suggests that for both newer and older devices, if the application requires less computation, then it is energy efficient to handle such a task locally. Nevertheless, it is always beneficial for larger and computation-intensive files to be processed under the thin-client scenario in order to lower the usage-phase energy and achieve a longer-usage lifetime.

## 4.8 Conclusion and Future work

To conclude, in this work, a thin-client approach for reutilization of smartphones. First an analytical model was developed to understand the imbalance in the life cycle energy due to short life span of smart mobile devices. A comprehensive numerical analysis was performed to calculate the average life cycle energy reduction due to reutilization of the discarded mobile phones. A reduction in the average LCE of approximately 11% was observed as a result of three months increase in the duration of usage phase. It was observed that with less usage stage duration, there exists a disproportion between the manufacturing energy and usage stage energy consumption. Thus, more sustainable devices can help achieve proper balance between the manufacturing energy and usage stage energy. Further, the thin client paradigm for reusing mobile

devices was empirically tested. The thin client paradigm effectively allows for a cloud based software approach so as to increase the number of usage phase by providing the required upgraded through the cloud.

The proposed model for reusing mobile phones under the thin-client paradigm was experimentally implemented by means of a lab-based setup of the thin-client scenario. During the empirical study, the performance and energy savings of reused mobile devices operating under thin-client paradigm were compared and analyzed with newer mobile devices having superior hardware capabilities. An Android-based application similar to that of typical word-search application was developed to perform word searches for a preselected word or phrase in the input word file. It was observed from the experiments that by increasing the file size from 10 KB to 75 KB, the computation time required for local processing on the newer device with better hardware specifications required approximately 90% more time than the older mobile device, which was reused under the thin-client paradigm. Also, it was realized that energy could only be saved on mobile devices under the thin-client scenario if the processing energy consumed on the local device was greater than the energy consumed by the wireless interface to process and send the data for remote processing. It was also observed that for the ~1 MB file size test case, approximately 85% of the energy could be saved under the thin-client-server paradigm for all devices.

In future work, a further case study could be performed to analyze the feasibility for reusing mobile devices under the thin-client paradigm for various purposes like classroom learning, research, and educational labs. Various educational applications and laboratory test-beds to extend the performance study done in this work could be developed. Thus, the results of this work not only provide insight on the impact of various parameters of reusing mobile devices with

a thin-client paradigm, but also provide future guidance to develop optimal approaches for energy-efficient application management in mobile devices when remote and local domains are available. This work is expected to serve as an initial step towards designing more sustainable and power-efficient mobile devices that improve user satisfaction through longer battery life, a reduction in the environmental footprint by reduced energy consumption, and a decrease in electronic waste.

# CHAPTER 5

# CONCLUSION

To conclude, this work proposed an optimal approach for energy-efficient application management in mobile devices when remote and local domains are available. This dissertation work provides an analysis of performance and energy consumption of mobile devices used under a cloud-based computing paradigm. This dissertation also analyzed the impact of cloud-based applications on the battery life of mobile devices and characterized the scenarios under which cloud-based applications would be relatively more energy-efficient for users of mobile devices. This dissertation investigated the factors that may affect the user experience and performance of any cloud application, as opposed to traditional non-cloud versions executed locally on a mobile device. Lastly, the issue of environmental sustainability was addressed by characterizing the overall life-cycle energy (LCE) consumed by mobile devices. This dissertation also proposed a possible thin-client approach for mobile devices to reduce each individual device's energy consumption (including the cost of energy to manufacture them) and the rate at which these devices are discarded.

# REFERENCES

# REFERENCES


[1] P. Somavat, S. Jadhav, and V. Namboodri, "Accounting for energy consumption of personal computing including portable devices," in *Proceedings of International Conference on Energy-Efficient Computing and Networking Archive*, 2010, pp. 141–149.

[2] California Department of Toxic Substances Control, "Cell Phone Recycling," 2007, available from World Wide Web: <http://www.dtsc.ca.gov/hazardouswaste/universalwaste/cellphonerecycle.cfm>.

[3] P. Somavat and V. Namboodiri, "Energy consumption of personal computing including portable communication devices," *Journal of Green Engineering*, vol. 1, no. 4, July 2011, pp. 447–475.

[4] P. E. Gobry, " ANALYST: Smartphone sales will dwarf pc sales this year and reach a staggering 1.5 billion per year by 2016," February 29, 2012, available from World Wide Web: <http://www.businessinsider.com/smartphone-sales-forecast-2012-2>.

[5] Mobi Thinking, "Global mobile statistics 2012," June 2012, available from World Wide Web: <http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/d#mobilebehavior>.

[6] T. Lively, "CELL PHONES: An overview of global mobile media accessibility," Communications Department, WVSU, First American Rights, 2011.

[7] N. Smith, "Battery life a major drain of smartphone satisfaction," March 15, 2012, availbale from World Wide Web: <http://www.businessnewsdaily.com/2200-smartphone-customer-satisfaction-battery-life.html>.

[8] E. M. Huang and K. N. Truong, "Breaking the disposable technology paradigm: opportunities for sustainable interaction design for mobile phones," in *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, ACM, April 5–10, 2008, pp. 323–332.

[9] United States Environmental Protection Agency, "Electronic waste (e-waste) recycling facts," 2005, available from World Wide Web: <http://www.ecrrecycling.com>.

[10] M. Khurrum, S. Bhutta, A. Omar, and X. Yang, "Electronic waste: A growing concern in today's environment," *Economics Research International*, vol. 2011, Article ID 474230, 2011, pp. 1-8.

[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2009-28m, February 2009.

[12] Flickr, 2013, available from World Wide Web: <http://www.flickr.com>.

[13] Picasa, 2013, available from World Wide Web: <http://picasa.google.com>.

[14] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of 2010 USENIX Annual Technical Conference*, June 23–25, 2010, pp. 1–12.

[15] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proceedings of 2nd USENIX Conference on Hot Topics in Cloud Computing (Hot Cloud'10)*, June 22–25, 2010, pp. 4.

[16] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proceedings of 2nd International ICST Conference on Mobile Computing, Applications, and Services*, Santa Clara, CA, October 25–28, 2010, pp. 59–79.

[17] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," *Proceedings of 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, ACM, June 15–18, 2010, pp. 49–62.

[18] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "SmartDiet: Offloading popular apps to save energy," in *Proceedings of SIGCOM*, ACM, August 13–17, 2012, pp. 297–298.

[19] K. Kumar and Y.H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, 2010, pp. 51–56.

[20] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems* (*EuroSys'11)*, April 10–13, 2011, pp. 301–314.

[21] W. Vereecken, L. Deboosere, P. Simoens, B. Vermeulen, D. Colle, C. Develder, M. Pickavet, B. Dhoedt, and P. Demeester. "Energy efficiency in thin-client solutions," in *Networks for Grid Applications*, Berlin, Heidelberg: Springer, 2010, pp. 109–116.

[22] T. Ghose, V. Namboodiri, and R. Pendse, "An analytical study of power consumption in portable thin-clients," in *Proceedings of IEEE Global Telecommunications Conference (IEEE GlobeCom)*, Houston, TX, December 5–9, 2011, pp. 1–5.

[23] Monsoon Solutions Inc., "Power Monitor," 2008, available from World Wide Web: <http://www.msoon.com/LabEquipment/PowerMonitor/>.

[24] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, October 28–28, 2010, pp. 105–114.

[25] PowerTutor, "A power monitor for andriod based mobile platforms," October 9, 2011, available from World Wide Web: <http://ziyang.eecs.umich.edu/projects/powertutor/>.

[26] M. Gargenta, *Learning Android: Building Applications for the Android Market*, Sebastopol: O'Reilly Media, 2011.

[27] u-Test, "The essential guide to mobile app testing," 2013, available from WorldWide Web: <http://www.utest.com/landing-blog/essential-guide-mobile-app-testing>.

[28] Developers, "Android application development," 2013 available from World Wide Web: <http://developer.android.com/training/basics/firstapp/index.html>.

[29] K. Wac, S. Ickin, J. Hong, L. Janowski, M. Fiedler, and A. K. Dey, "Studying the experience of mobile applications used in different contexts of daily life," *Proceedings of First ACM SIGCOMM Workshop on Measurements up the Stack*, August 15–19, 2011, pp. 7–12.

[30] J. Huang, Q. Xu, and B. Tiwana, "Anatomizing application performance differences on smartphones," in *Proceedings of 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, ACM, June 15–18, 2010, pp. 156–178.

[31] A. P. Dinda, G. Memik, R. P. Dick, B. Lin, A. Mallik, A. Gupta, and S. Rossoff, "The user in experimental computer systems research," in *Proceedings of 2007 Workshop on Experimental Computer Science*, ACM , June 13–14, 2007, pp. 10–10.

[32] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 02–05, 2002, pp. 217–226.

[33] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proceedings of 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '01)*, 2001, pp. 238–246.

[34] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, September 2004, pp. 795–809.

[35] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *IEEE Computer*, vol. 43, no. 4, 2010, pp. 51–56.

[36] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of Sixth Conference on Computer Systems (EuroSys'11)*, April 10–13, 2011, pp. 301–314.

[37] V. Namboodiri and T. Ghose, "To cloud or not to cloud: A mobile device perspective on energy consumption of applications," in *Proceedings of IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (IEEE WoWMoM)*, San Francisco, CA, June 25–28, 2012, pp. 1–9.

[38] V. Namboodiri and T. Ghose, "Cloud version or local verson? A mobile device perspective on energy consumed for application execution," *Journal of Pervasive Mobile Computing*, 2013 (in review).

[39] T. Ghose, V. Namboodiri, and R. Pendse, "Energy dependent analytical model for sustainable portable computing devices," in *Proceedings of IEEE International Symposium on Sustainable Systems and Technology (IEEE ISSST)*, Boston, MA, May 16–18, 2012, pp. 1.

[40] T. Ghose, S. S. Argade V. Namboodiri, and R. Pendse, "Efficient Resource Managemant for Sustainable Mobile Computing," *International Symposium on Sustainable Systems and Technology ( ISSST)*, Cincinnati, OH, May 15–17, 2013, pp. 1-7.

[41] Nokia, "Each and every Nokia device is created with the environment in mind," 2013, available from World Wide Web: <http://www.nokia.com/environment/devices-and-services/creating-our-products/environmental-impact>.

[42] S. E. Hanselman and M. Pegah, "The wild wild waste: e-Waste," in *Proceedings of the 35t annual ACM* (*SIGUCCS '07)*, October 7–10, 2007, pp. 157–162.

[43] International Telecommunication Union, "Techwatch alert on batteries for portable ICT devices," February 2010, available from World Wide Web: <http://www.itu.int/dms_pub/itu-t/oth/23/01/T230100000E0001PDFE.pdf>.

[44] M. Etoh, T. Ohya, and Y. Nakayama, "Energy consumption issues on mobile network systems," *International Symposium on Applications and the Internet (SAINT 2008)*, July 28–August 1, 2008, pp. 365–368.

[45] G. Hwang, "Supporting cloud computing in thin-client/server computing model," in *Proceedings of IEEE 2010 International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, September 6–9, 2010, pp. 612–618.

[46] In-Stat: An NPD Group Company, 2000, available from World Wide Web: <http://www.instat.com/>.

[47] E. Davis, "Green benefits put thin-client computing back on the desktop hardware agenda," White Paper, Forrester Research Inc., March 10, 2008.

[48] EDUCAUSE, "What are thin-clients," 2010 available from World Wide Web: <http://net.educause.edu>.

[49] T. Ghose, V. Namboodiri, and R. Pendse, "An evaluation of thin-client solutions for sustainable mobile computing," *Journal of Computers and Electrical Engineering*, 2013 (in Review)