1-1-2016

System Support For Energy Efficient Mobile Computing

Youhuizi Li
Wayne State University,

SYSTEM SUPPORT FOR ENERGY EFFICIENT MOBILE COMPUTING

by

YOUHUIZI LI

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2016

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my beloved family.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to those who supported and encouraged me in one way or another during the last five years.

First of all, I would like to give the deepest gratitude to my advisor, Dr. Weisong Shi, who has provided me with valuable guidance and endless patience. Dr. Shi taught me how to find interesting research problems and how to come out solutions step by step. During tough times in the Ph.D. pursuit, I faced huge challenges and couldn't move on. Dr. Shi always kindly gave me suggestions and encouraged me to keep working. He is not only a knowledgeable professor in academia, but also a kind and wise elder who impacts my life a lot. I cannot succeed without his support and guidance.

I also want to extend my thanks to Dr. Nathan Fisher, Dr. Caisheng Wang and Dr. Hongwei Zhang for serving as my committee members. Their professional suggestions on the prospectus are very valuable for me to improve my work. They also showed me how to be an excellent researcher and educator.

Moreover, I would like to thank former and present MIST and LAST group members that I have had the pleasure to work with or alongside of. Especially for Shinan and Hui, who mentored me in this field, and Pradeep, who assisted me to setup and use the ThermoStream device. With these brilliant people, we had many exciting discussions in the laboratory.

Last but not least, I deeply appreciate the support, encourage and love from my mother Chunfang You and my father Xinhua Li. They have been a constant source of strength and always there for me through the good times and bad times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Mobile devices are developed rapidly, from PDAs (personal digital assistant) to current smartphones and wearable devices. They greatly improve our daily life and make things easier and convenient. However, the battery drain problem is a critical issue that hurts user experience and restricts device's functionality. Previous researchers mainly focused on reducing power dissipation of hardware components and avoiding software energy bugs, they did not successfully satisfy the users' battery usage demand. In this dissertation, we propose system support for energy efficient mobile computing, which includes user behavior learning, online power profiler, energy efficient runtime scheduling and energy-based thermal control policy. All of the modules work together can greatly improve system energy efficiency and extend battery life.

1.1 Motivation

Mobile computing has been an integrated part of our daily lives and will be more involved in the future. Mobile device serves as our wallet, our notebook, our timer and our most important tool to connect with people. The number of industries and activities being transformed by mobile will only continue to grow [1]. The number of Android applications already over 1.8 million [2]. The application downloads in the App Store has passed 100 billion as announced in the Apple Worldwide Developer Conference this year [3] and the predicted downloads will over 250 billion in 2017 [4]. We leverage modern mobile devices to do all kinds of things, such as monitoring health status and activity data from smartband, augmented reality from HoloLens [5] and so on. In addition, the emerging of Internet of Things (IoT) will greatly change our living environment. Smart home, smart building and smart traffic management system start to step into our daily life. There will be 25 billion permanently connected things (objects that can connect to the internet) and 200 billion intermittently connected things by 2020, the connected device in households will be doubled [6]. Based on a report presented by Business Insider [7], the number of IoT devices will increase with a compound annual growth rate of 35% from 2015 to 2019.

The huge number of mobile devices and users shows the fact that mobile computing really improves people's life. To provide better service, develops and vendors work hard on every aspect. From processor aspect, engineers employ several techniques, such as increasing core operating frequency and voltage, using larger on-die caches, leveraging symmetrical multiprocessing and heterogeneous multi-core computing [8, 9]. These technologies are applied by most manufacturers. On the wireless communication aspect, which builds the foundation of mobile computing, the faster network speed is always wanted. Compared to 75 Mbps of 4G LTE service, Samsung has achieved 5G speeds in the range of 1 Gbps in the lab, and the company expects that the technology could eventually provide speeds in the tens of gigabits per second [10]. Moreover, modern smartphones are equipped with high-end cameras which can capture Quad HD video, more sensors that monitors your all day activity, and bigger screen in a lighter body to provide users better performance.

All of the high performance functionality requires more power support. However, a battery for a thin mobile device cannot provide enough energy. The battery life of Google Glass only lasts about three hours [1]. The battery drain issue is the single main gripe of today's mobile phone user [11]. 33% users choose better battery life as the most wanted feature [12]. Battery life seriously affects the user experience and stops them enjoying the mobility. Many places, like airports and restaurants, provide power sources for clients to charge their devices. One of the main reasons is the slow development of battery, it is far behind the demands. Take iPhone as an example, battery capacities of the various models have grown about 15% since its introduction while computing power grows exponentially [13].

Several approaches are proposed to extend battery life. From hardware's perspective, recent hardware components support different power states so that they can stay in the low power state when they are not busy [14]. As the diversity of the mobile applications, heterogeneous architecture (CPU-GPU [15], CPU-DSP [16], ARM big.LITTLE) is applied to satisfy different levels of requirements so that we can achieve better performance and high energy efficiency.

From operating system viewpoint, Advanced Configuration and Power Interface (ACPI) [17] are commonly applied to manage the power usage. From the simple way of setting screen off time to complicated methods, like bundle IO activity [18] which delays short transfers and batches them with delay-sensitive transfers [19], users try every way to increase battery life. However, the battery drain issue is still not well solved.

In addition, due to the small size and high power density of mobile devices, thermal management also becomes the design bottleneck [20, 21, 22, 23, 24]. Motivated by the close relationship of thermal and power, we should also consider the thermal effects when improving system energy efficiency. For instance, we cannot put the CPU on the highest frequency because the high power may cause temperature overs the predefined critical threshold, although the running time can be greatly reduced which leads to less energy consumption.

Since the battery resource is very limited, we need to pay more attention to how to use it efficiently. Figure 1.1 describes a general process of what happens when a user plays with a mobile device. First, the user interacts with applications and select one to run. Then, the operating system schedules the application and allocates its required resources which include CPU, memory, network and so on. When the application is running, the generated heat will influence the device's temperature and the thermal management system should take action. To save energy in such a typical scenario, the first focus should be the user behavior. *Which application the user plays? How long it lasts? Are there any patterns can be found? How much energy saving space is available?* Mobile devices are developed to serve humans, and the usability should always be the first priority. With the user's preference, we can put more effort on the "frequently used" parts. Besides, the application's power behavior should be analyzed so that we know *where does the power go* and *if the hardware is used properly and efficiently*. Take advantage of the system's characteristics, we can propose energy-motivated scheduling and thermal management in operating system level. Since users care about the energy consumption of whole device, particular applications normally do not have as much influence as system

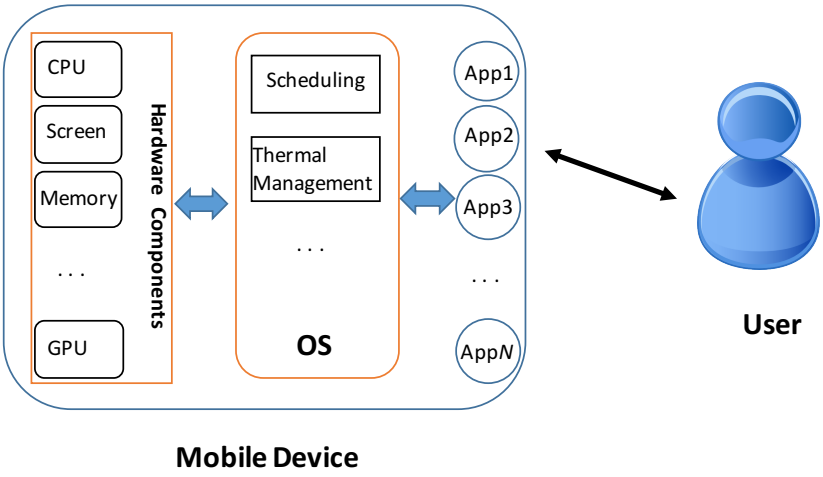


Figure 1.1: The overview of a general user-device interaction scenario.

services. In a word, single optimization approach cannot reach the optimal results. All the modules should work together to eventually improve the system energy efficiency and extend battery life.

1.2 Objectives

Given the challenges and opportunities discussed above, this dissertation aims to provide system support for energy efficient mobile computing so that we can systematically improve the energy efficiency of each part along the process and cooperate them as a whole to achieve the optimal energy saving.

Specifically, this dissertation will accomplish the following objectives:

1. With smartphone as an example of mobile device, analyze the user interaction behavior to figure out the potential energy saving space and usage patterns which provide directions to improve system energy efficiency and user experience.
2. Develop a low overhead power profiling tool to obtain the power dissipation information of applications and systems so that we know where does the power go and how to evaluate the energy difference.
3. Take heterogeneous platform as an example, analyze the influence of system features on energy consumption and take advantages from them to improve energy efficiency.

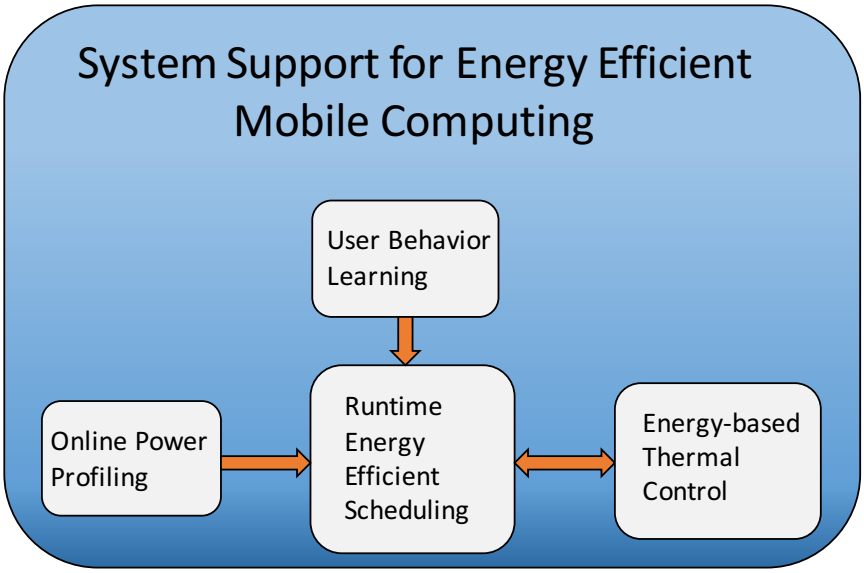


Figure 1.2: The overview of our approaches.

4. With fan as an exemplary cooling method, develop an energy-based thermal control policy.

1.3 Our Approach

As presented in Figure 1.2, our system support includes four main components: user behavior learning, online power profiling, runtime energy efficient scheduling and energy-based thermal control. User behavior learning module monitors user-device interactions and analyzes usage patterns. After that, the potential battery extended time can be estimated and we know more about users' preference which helps improve user experience and provides energy saving directions. For example, the system can schedule or suspend proper processes based on usage patterns. To improve energy efficiency, we first need to figure out where does the power go. Online power profiler collects different hardware components' status to calculate system and application level power dissipation. As heterogeneous platforms become the trend of the future mobile devices, the scheduling module decides which hardware component the system will use, e.g. which CPU core to use in the big.LITTLE platform. The power information can be treated as an important factor for system to schedule applications to the right processor so that we can actually leverage the low power feature of heterogeneous platforms. In addition,

high temperature is another key issue for mobile devices, which is closely related to power dissipation. On one hand, the cooling method fan itself needs to consume amount of energy that may influence system energy efficiency. On the other hand, to maintain the thermal constraint, thermal management system can restrict some of the system functions which include scheduling workload to a low frequency core. Its strategy directly affects the system's power behavior. In the following, we will generally describe each part.

1.3.1 User Behavior Learning

Mobile devices have become an integral part of our daily life, people use them to check emails, watch videos, send instant messages, and so on. There are several previous works [25, 26, 27, 28] that studied the user behavior from application usage perspective. Generally speaking, the application usage is diversity, which is presented on interactions per day, data received/sent per day, most popular applications and so on. The diversity directly contributes to the different battery life for different users.

Understand user interaction behavior is very important. On one hand, it helps developers build more "smart" applications and services which take further action before user asks, for instance, pre-loading the applications user will use. On the other hand, it points the direction to save energy from user behavior aspect. For example, turning off the unimportant applications when the battery is low so that the basic functions like making a phone call and receiving/sending a message can work. Based on the diversity of the usage of applications, the same energy saving method may have different results for different users. The approach that increase GPU energy efficiency gets better results from users who play games a lot than users that prefer using office applications.

In this part of the dissertation, we built a model to predict the battery life, which considers the influence of both user behavior and hardware components. Assuming the application power is relatively stable and the user behavior pattern is known, we can analyze the influence of each hardware component on the device's battery life. The model provides a mechanism to evaluate

various energy saving methods for different users. We verified the assumption conditions by analyzing the application power and logged user behavior data. At last, several aspects that may affect prediction results are discussed, for instance, the sleep frequency will cause the "tail energy" overhead for some users.

1.3.2 Power Profiling on Mobile Devices

To improve the energy efficiency, the first step is to figure out *where does the power go*. There are two categories of power profiling approaches: hardware-based power profiling and software-based power profiling. The straightforward method is attaching power meters or power sensors to directly measure the power dissipation. However, hardware-based power profiling needs extra components to measure the power data, which is not convenient to use on mobile devices and the built-in power sensors are not always supported. Hence, researchers are more interested in software-based power profiling. Basically, the software-based method calculates the power according to a series of power models. The accuracy of the power models depends on the power indicators of the model and the way we get the indicators' value. Asides from the accuracy, another important metric of profiling is the overhead. Our goal is to save the energy, so the profiler itself cannot be a power hungry process. Besides, the effects to the system performance should also be considered since we analyze the power data online and do not want to delay other applications which may hurt user experience.

Both the system level and application level power information are critical to saving devices' energy. System level power information presents the overview of the energy consumption. It directly influences the battery life. The system level power information is one of the standards to evaluate the performance of different energy saving approaches. For application level power information, it is the key to figure out the underlying reasons of energy wasting. Software is the biggest energy consumer, all the resources are responsible to satisfy its demands. Some applications abuse the resources (e.g. CPU) and lead to the waste of energy, which also explains why the energy efficiency of the applications with the same functionality are different.

Moreover, some of them even become energy hungry malwares. System level power profiling is hard to find out the specific power hungry applications as well as which resources the application overused. Hence, the system level power information and application level power information are both needed to be considered.

In these situations, we designed and built a power profiler, Bugu, to understand power behavior of mobile systems. It supports application level and system level power profiling as well as detailed component usage information includes CPU, memory, GPS, sensors, etc. Besides, it monitors system events, such as wakelock requiring and releasing, Wi-Fi on and off, so that these resource abuse can be easily detected. We provide comparing service for applications with the same functionality which helps users to design and choose more energy efficient applications. Moreover, in this dissertation, we analyzed 100 popular applications' power behavior and revealed the root causes of high power consumption for some of them. From the result we observed several aspects that can be improved to save both application and system energy, such as sensors and video module energy efficiency, phone service *rild*.

1.3.3 Runtime Energy Efficient Scheduling

Heterogeneous platforms usually appear in server clusters [29, 30, 31, 32, 33] as the hardware configuration and performance of new servers are different with old servers. With more and more researchers work in this field, the heterogeneity is introduced to multi-core architecture area to improve performance and power efficiency. Kumar *et al.* [34] proposed the potential power reduction on their single-ISA heterogeneous multi-core architectures during an application's execution. It built the foundations for the following energy aware task scheduling research [35, 36, 37]. The advantages of heterogeneity on performance increasing and power reduction bring it into the mobile field. Some researchers proposed to move part of the computational work to DSP or GPU [16, 15, 38], some people preferred to offload tasks to the cloud [39, 40]. ARM big.LITTLE processor enhanced its position in the mobile area. The heterogeneity is the trend of the coming future. Hence, we propose energy efficient scheduling

as one of the system supports which focuses on the heterogeneous platform.

The first step to take the advantage of heterogeneity is runtime migration and scheduling. There are several choices in front of us, the critical problem is using the right part at the right time. Take the big.LITTLE platform as an example, the issues we need to solve include *which core is the proper one to run the applications, when is the right time to change the core from the big (little) core to the little (big) core* and *how to migrate the current workload.*

To solve these problems, we first compared the heterogeneous platform with homogeneous platform on the aspects of performance and energy to figure out the benefits we may get. Then we evaluated the overhead of migration and penalty of wrong scheduling. We found that heterogeneous platform indeed has great potential for energy saving, but the penalty for wrong scheduling is also high (up to 30% more energy). There are several aspects need to be improved to leverage the heterogeneity, such as fine-granularity power control and thread level parallelism.

1.3.4 Energy-based Thermal Control

The reliability of electronic hardware components is closely related to their operation temperature. For example, the processor failure rate *exponentially* depends on the temperature [41]. Thermal control normally is a critical issue for data centers as they need to guarantee performance and reduce the cooling cost at the same time. Several approaches were proposed, such as phase change materials [42] and chilled water tanks [43, 44, 45], to absorb the generated heat. As the mobile devices become more and more powerful, the high performance and power density also lead to thermal issue. Currently, when the temperature reaches the trip point, the system performance will greatly decrease as a result of thermal throttling or the device will directly shut down.

To avoid hurting the performance a lot, we can apply a cooling method to dissipate heat. We take fan as an exemplary cooling method since it is commonly used in computers. Comparing with phase change materials that require hardware redesign, the cost of fan is low. We can

experimentally analyze its cooling performance and other influence on the system.

Low power is the advantage of SoC platforms, however, the power of fan can reach 40% (0.8 W) of the experimental board (Odroid-XU+E [46])'s idle power when the fan runs at full speed. Although we need to control the temperature, we also cannot sacrifice energy efficiency. In addition, comparing with servers which stay in an air conditional room, the devices involved in mobile computing mostly in outdoor environments, such as IoT gateways. So, the thermal control policy should adapt to various environments. Hence, we built a thermal prediction model that considers components' power dissipation, temperature history and the ambient temperature. Based on that, we proposed an energy-based thermal control policy Falcon. It is a hybrid approach. On one hand, it uses fan as active cooling method and proactively tunes the fan speed to satisfy the thermal constrain with low power dissipation in room temperature or lower. On the other hand, it adjusts CPU frequency to restrict the generated heat in high ambient temperature environment.

1.4 Contributions

The contributions of this dissertation include:

1. We monitored smartphone user interaction behavior data, analyzed the usage patterns and classified the users into six user types. Based on the usage information, we proposed a prediction model that estimates battery life according to user behavior and hardware usage. We studied the influence of user interaction habit on the energy consumption of mobile devices. The potential battery life can be extended from 1.03 to 2.24 times for different user type.
2. We developed a power profiler Bugu which provides both system and application level power information. Our experiments show that Bugu is accurate enough (95%) with a low overhead (2.52% of foreground application's power on average). In addition, we analyzed the power behavior of 100 popular mobile applications on different platforms. The energy consumption of applications with same functionality varies a lot. Several

implications are derived based on the observations. For examples, some power management APIs (like wakelock) are not used properly, which leads to huge amount of energy waste. Radio service and interrupts generated by sensors cause system cannot enter the deep sleep state.

3. We investigated the ARM big.LITTLE heterogeneous platforms. Comparing with homogeneous platforms, most of the energy savings in heterogeneous platforms come from idle time and there is no much benefit for sustained heavy workloads. We analyzed the impact of scheduling and migration on the ARM big.LITTLE devices from the performance and energy aspects. The improper scheduling can consume 5% - 30% more energy. We derived a list of insights to fully leverage heterogeneity, such as fine-granularity power control and thread level parallelism, related to hardware, application and operating system design.
4. We built a runtime thermal prediction model based on temperature history and components' power dissipation. We proposed Falcon, an ambient temperature aware fan control policy for smart gateways. Take Odroid-XU+E as an example platform, we experimentally showed that Falcon can save 4.85% total system power and reduce noise in 34% of the time on average. In addition, the two environment adaptive mechanisms applied in Falcon help us achieve the same thermal control ability in different ambient temperature environment.

1.5 Outline

The rest of this dissertation is organized as follows: Chapter 2 reviews the related work. Chapter 3 analyzes the user device interaction behavior and presents the battery life prediction model based on the user behavior and hardware usage information. It estimates the potential battery savings we can achieve if users change their behavior a little bit as well as the influence of hardware improvement. Chapter 4 introduces the power profiler *Bugu* which illustrates where the power goes for both applications and systems. The power behavior analysis of

100 applications indicates several energy saving directions. To figure out the energy influence of system features and the advance in modern hardware architecture, Chapter 5 describes the comparison of homogeneous platform and ARM big.LITTLE heterogeneous platform. The experimental results reveal the importance of the scheduling and opportunities to improve energy efficiency from heterogeneity. Following that, Chapter 6 takes fan as an exemplary cooling method, discusses the energy cost of thermal control and demonstrates an energy-based control policy Falcon. Finally, Chapter 7 concludes this dissertation and Chapter 8 presents the future work.

CHAPTER 2 RELATED WORK

As described in the last chapter, we aim at providing system support for energy efficient mobile computing. In this chapter, we describe related work in several areas: user behavior analysis, power profiling, energy saving approaches and thermal control management.

2.1 User Behavior Analysis

User behavior is very important for developers, it reveals more information about users and points the directions that improve the user experience. Several previous works [25, 26, 27, 28] studied the user behavior from application usage behavior perspective. Yan *et al.* [47] leveraged context information to pre-launching applications which improves the user experience. Huang *et al.* [48] proposed the framework that can intelligently control home devices based on the history of user behavior.

In addition to improve performance and user experience, user behavior data is also valuable in energy saving field. Falaki *et al.* [25] collected two user groups' information to analyze the usage of the smartphones. The heaviest users drain close to 250 mAh, while the lightest of users drain only 10 mAh. The huge energy consumption difference is caused by different usage habit. Shen *et al.* [16] proposed an energy efficient video downloading strategy which is determined progressively during the playback process with the consideration of user demand that is predicted from user behavior.

Generally speaking, different people have different preference and their habit can be illustrated in threefold. First, their favorite applications are different. For example, businessmen like checking the stocks all the time, while students prefer refresh social network applications in case they miss some news of their friends. Second, same person prefers different applications based on the time and location information. Usually we open work related applications, like email, more frequently on Monday to Friday, while we prefer to play with entertainment related applications, such as games and video players, in the weekends. Similar situations for the workplace and home, which is classified from the location perspective. Third, users have different interaction habit, even for the same applications. Take Facebook as an example, some

people like refreshing news to know others' situation, while some users prefer post their status.

2.2 Power Profiling

To save the energy, the first step is figuring out how much energy it consumes. As the foundation of energy/power related research, there are a lot of power profiling approaches are proposed. These approaches can be classified to two categories: hardware-based power profiling and software-based power profiling.

Using hardware to measure the current and voltage is the straightforward way to get power data. It is also commonly used to verify the accuracy of software-based profiling. The prototype version of PowerScope [49] uses a digital multi-meter to sample the current drawn of the profiling computer and records system activity at the same time. As a result, it generates an energy profile for later analysis. PowerPack [50] also uses digital meter to measure, while it focuses on each hardware component (CPU, disk, memory, etc.). Joseph et al. [51, 52] measure the power dissipation on a high performance processor . They tested a bunch of microbenchmarks to evaluate performance and power tradeoff. Normally, a system current is calculated through the resistor. Most mobile devices are measured directly by the power meters, such as Monsoon power meter [53] and BK Precision power supply [54]. The power meter is the power supply for the device, it provides stable voltage and records the current. Most software-based power profiling methods use power meters to verify the accuracy [55, 56, 57, 58], so as our Bugu. Moreover, there are some servers that have built-in sensors that can directly get power information [59, 60], while the power sensors are not implemented on mobile devices.

Compare with hardware-based power profiling which needs extra devices/sensors to measure the power consumption, software-based power profiling is more convenient and flexible. Dempsey [61] extracts power parameters to model the disk drive power consumption. Bertran *et al.* [62] took advantage of performance counters to build power models and provided per component power consumption. Quanto[63] system addresses network communication power model in embedded system, the key parameter is network event. Dong and Zhong [64] pro-

vided a self-constructive approach to build system energy model for mobile systems by using the smart battery interface to get enough information. Generally speaking, the critical part of software-base power profiling is power models. System level and application level power models are built according to different power states of hardware components, the trigger of the state change and the approach to get the trigger information. Zhang *et al.* [55] collected power traces for hardware components and built power meter based and battery based power models. Pathak *et al.* [58] proposed system-call-based fine-grained power estimation by modeling power state using Finite State Machines (FSM). Yoon *et al.* [65] monitored kernel activities of hardware component requests to get usage statistics for component models. Furthermore, the software-based power profiling can be applied to fine-grained level. Wang *et al.* [66] proposed Safari which provides function-level power information. Li *et al.* [67] took advantage of program analysis and statistical modeling to calculate source line level energy information.

2.3 Energy Saving Approaches

After given a brief introduction of power measurement and profiling, we will discuss existing energy saving approaches in this section. As we all know, energy is becoming a bottleneck for mobile devices. Researchers proposed different energy saving approaches from system level and component level.

As early as 2002, energy has been treated as a first-class resource in ECOSystem [68] which contains a currentcy model and allocate the energy to different tasks according to user preferences to extend battery lifetime. Besides, Koala [69] predicts the performance and energy consumption and dynamically control frequency to save energy. In mobile field, researchers also did a lot of work to improve energy efficiency so that the battery life can be extended [70, 71]. For example, Cinder [72] also leverage the idea and treat energy as a resource, but it allocates energy directly to each process and uses a hierarchical structure to control the resource, which avoiding the competing between parent and child processes. Roy *et al.* [73] implemented Cinder and showed the good performance even with malicious applications.

Aside from modifying operating system to decrease energy consumption, some researchers pay more attention to specific components. EnTracked [74] uses dynamically changing context to schedule GPS so that it is energy-efficient as well as keep the performance. MAUI [75] saves energy through fine-grained code offload after evaluating energy consumption under connectivity constraints. Duan and Bi [76] proposed a hybrid approach which leverages mobile RAM and phase change memory to achieve memory energy optimization. Pathak *et al.* [18] presented an energy accounting approach on application level and proposed saving energy by optimizing I/O bundle. Pering *et al.* [77] proposed CoolSpots which helps mobile devices to automatically switch between multiple radio interfaces, such as WiFi and Bluetooth to improve communication energy efficiency. Rozner *et al.* [78] leveraged access point virtualization and developed a fair scheduling algorithm that reduces retransmissions so that they can save the Wi-Fi energy of mobile side. Priyantha *et al.* [79] focused on continuous sensing applications, they developed a sensing architecture to offload part of sampling and processing work to the a low power processor.

Heterogeneous platform appears in clusters, high performance multi-core computers. With the increasing demanding of mobile computing, it is the trend for mobile platforms so that they can provide high performance with low power dissipation. Kumar *et al.* [34] proposed single-ISA heterogeneous multi-core architecture and discussed the potential power saving. Awan and Petters [35] proposed a energy-aware partitioning of tasks on a heterogeneous multicore system, which saves energy through choosing the proper sleep states of cores. For mobile field, Lin *et al.* [80] designed and implemented a operating system for heterogeneous coherence domains. It has two kernels running on top of the two coherence domains and it can greatly improve the energy efficiency of light OS workloads. Carroll and Heiser [81, 82] build a Linux frequency governor medusa which leverages DVFS and offlining to save energy of mobile system. Zhu and Reddi [83] improved performance and energy efficiency of mobile web browsing on the big/little system by estimating webpage load time and energy consumption

and scheduling webpages to proper core and frequency.

2.4 Thermal Control Management

As the power density becomes more and more higher, thermal issue is treated as one of the system design bottlenecks. The reliability of hardware components are greatly affected by operating temperature. For example, the processor failure rate *exponentially* depends on the temperature [41]. Researchers are working hard to solve the thermal issue. Generally speaking, there are two kinds of approaches: restricting heat generation and acceleration heat dissipation.

From the heat generation viewpoint, several papers [84, 85, 86, 87, 88] discussed the influence of task scheduling that dynamically turns on/off servers based on workload behavior and cooling efficiency, balances the workload to avoid local hot spot. In the aspect of cooling approaches, phase change materials [42], chilled water tanks [43, 44, 45] are proposed to absorb the generated heat during workload executing. In addition, free cooling [89, 90] is also applied to reduce energy cost in many geographical locations.

Due to the nature of SoC platforms, workload scheduling and phrase change materials are not applicable. ARM developed Intelligent Power Allocation thermal framework [91] which estimates power budget based on current and reference temperature and allocates it to CPU and GPU based on performance requirements. Kim *et al.* [92] proposed migration-based dynamic thermal management for heterogeneous processors that migrates applications to the little cores instead of decreasing frequency of the big cores. However, the drawback is the performance loss. Hence, we propose Falcon, a hybrid method, which only adjusts frequency when necessary.

CHAPTER 3 USER BEHAVIOR BASED BATTERY PREDICTION

For mobile devices, battery energy is the most precious resource. In the last decade, researchers have proposed various energy saving strategies from the aspect of system and particular hardware components. In this chapter, we explored how the user interaction behavior influences the battery usage. We first developed a battery lifetime prediction model that considers the influence of both user behavior and hardware components. Through experiments we analyzed the assumptions and the accuracy of the prediction model. To analyze the impact generated by user behavior, we classified users into six types based on their application usage pattern. The theoretical battery life and potential extended battery time for each user type, with and without hardware improvement, have been illustrated.

The remainder of the chapter is organized as follows: Section 3.1 presents the motivation and overview of our work. The prediction model is described in Section 3.2. Then applications' power, user traces and model accuracy are analyzed in Section 3.3. Section 3.4 applies the model to each user category and predicts the battery life based on hardware improvement.

3.1 Introduction

Mobile devices are becoming more and more intrinsic in our daily lives, and global smartphone users will reach 1.75 billion in 2014, which is roughly one quarter of the world's population, and more than half of the population in the United States have their own mobile devices now [93]. However, the battery drain issue seriously influences the user experience and a survey shows that battery life is the single main gripe of today's mobile phone user [11]. To solve this problem, researchers have been trying to find optimization approaches to extend battery life from system level [94, 73] and component level [76, 95, 56]. However, it is hard to evaluate the energy saving performance of these approaches. For instances, How to compare the influence of memory saving with GPS saving? How long the battery lifetime can be extended for each application after applying the approach? We are still missing a standard mechanism to evaluate the effectiveness of the saving approaches from the perspective of real device usage.

Asides from saving energy from the device itself, user behavior also affects battery dis-

charging time seriously. Falaki *et al.* [25] collected two user groups' information to analyze the usage of the smartphones. They calculated the mean and standard deviation of energy that users drain in an hour. The results show that the battery usage is very different from user to user. The heaviest users drain close to 250 mAh, while the lightest of users drain only 10 mAh. Hence, users can extend the battery lifetime by adjusting their behavior although they may not like to. Moreover, the performance of energy saving approaches mentioned above are different for each user. As a result, the user behavior needs to be considered as an important factor when we improve battery life of mobile devices.

We undertake the following question: how close can one battery charge survive seven days for normal smart phone users. We first developed a prediction model that calculates how long the battery can be extended under various situations. The model takes both hardware components information and user behavior into consideration, providing a mechanism to evaluate various energy saving methods for different users. Assuming the application power is relatively stable and the user behavior pattern is known, we can analyze the influence of each hardware component to the device battery life. For some users, if the energy efficiency of the display is doubled, the battery lifetime will increase by 18.57%. From the users' point of view, the possible maximum battery lifetime can be calculated as well. For example, compared with the original 66h for users who rarely use their smartphone, we found that the battery life can be extended to 147h (more than 6 days) when we only maintain applications in the top three commonly used categories. Moreover, given a target battery discharging time, the prediction model will provide the information for how much improvement we need to achieve.

3.2 Battery LifeTime Prediction Model

In this section, we listed the assumptions that our prediction model is based on. Following that, we illustrated the power models that estimate average application power and the battery lifetime prediction model.

3.2.1 Assumptions

To simplify the problem, the applications and devices that we focus on should satisfy the following assumptions:

1. The average power of each hardware component is relatively stable and is linear to the usage of the component.
2. User behavior has specific patterns and the pattern can be expressed by the user-interacted applications' running time.

3.2.2 Lifetime Prediction Model

To calculate the battery lifetime, it is important to estimate the average power. Since the user behavior is represented by the application running time, the power should be calculated at application level. Table 3.1 presents the power models for the average power of the four main hardware components that we used to estimate application power. They are abstract formats used to show that the power is linear to the average component's usage. For CPU power, the main indicator is the utilization which is calculated from user time, kernel time and sampling interval. Since the power variation for different color is not very large for Google Nexus 4, we only consider brightness in display power. The radio and Wi-Fi power is based on the signal states and packets rate respectively. More details of the specific power models can be found in [96]. In the average power models, all c are constants and u are the usage of each component.

Component	**Average Power Model**
CPU	$m_{cpu} = c_{cpu} * u_{cpu} + c_{cpuidle}$
Display	$m_{display} = c_{display} * u_{display} + c_{displayidle}$
Radio	$m_{radio} = c_{radioscan} * u_{rs} + c_{radioon} * u_{ro} + c_{radioidle} * u_{ri} + c_{radioactive}$
Wi-Fi	$m_{wifi} = c_{wifilow} * u_{low} + c_{wifihigh} * u_{high} + c_{wifiidle}$

Table 3.1: The power models for main hardware components.

We denote

$$\vec{M} = (m_{cpu}, m_{display}, m_{radio}, m_{wifi})$$

as component power vector of one application, then the whole average power of the application is $a = sum(\vec{M})$. Now, suppose there are n applications, the average power of the ith application is a_i. Let

$$\vec{A} = (a_1, a_2, ..., a_n)$$

be the application power vector, and let

$$\vec{B} = (b_1, b_2, ..., b_n)$$

be the percentage of time that application is used by a user, then the average power for this user is

$$p = \vec{A} * \vec{B}^T.$$

Further, suppose the battery energy is E, then the life time is:

$$t = \frac{E}{\vec{A} * \vec{B}^T} \tag{3.1}$$

If the energy efficiency of the ith component can increase x_i times, the battery energy becomes E'. Let $s_i = \frac{1}{x_i}$, and let

$$\vec{M'} = (s_1 * m_{cpu}, s_2 * m_{display}, s_3 * m_{radio}, s_4 * m_{wifi})$$

be the new component power vector and let $\vec{A'}$ be the new application power vector, then the whole life time can increase

$$r = \frac{t_{improved}}{t_{original}} = \frac{E' * \vec{A} * \vec{B}^T}{E * \vec{A'} * \vec{B}^T} \tag{3.2}$$

3.3 Model Analysis

In this section, we analyzed the assumptions of the prediction model and presented the relationship of application power estimation error rate and lifetime prediction model accuracy.

3.3.1 Experiment Setup

For power related experiments, we set up the measuring environment as Figure 3.1 shows. The mobile device is a Google Nexus 4, and its specification is listed in Table 6.1. We use a

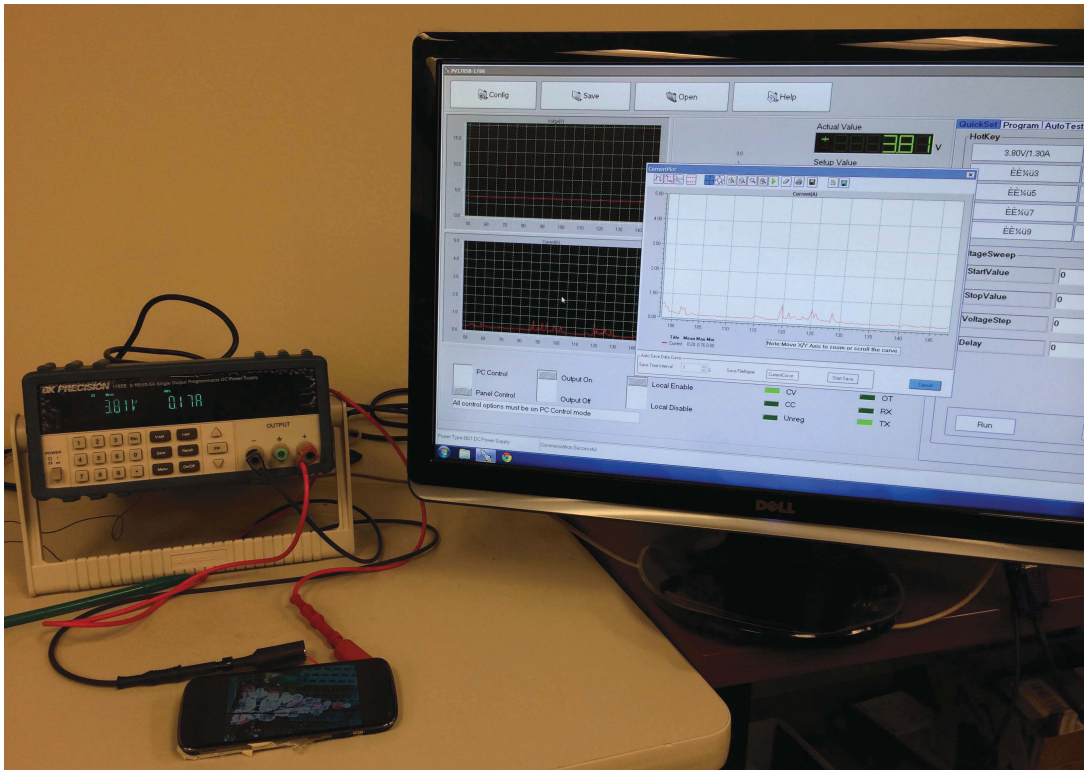


Figure 3.1: The experiment platform.

BK Precision programmable power supply [54] to power up the smartphone, which provides a constant voltage of $3.8V$ and a maximum current of $3A$. It samples the current four times per second, and sends the data to the computer. To reduce the interference, we installed the target application on a clear OS and stopped all unnecessary services, such as Google Plus and Google Play services. Thus, the device power fluctuation is mainly caused by the target application. Table 4.3 lists the applications used in the rest of the chapter. Some of them are among the top 10 most popular apps in Android Market.

For user behavior monitoring, we modified the *ActivityStack* class in the Android OS to log the *Pause* and *Resume* state with the time of each activity to get the application usage information [97]. In addition, the battery capacity was recorded when each time the screen was off. It presented the total energy consumption for the interactive session. We gave two Google Nexus 4 phones to 14 students who were randomly picked throughout the university. The device was their primary smartphone during the one week experiment time, they were free

Component	Specification
OS	Android 4.3; kernel version 3.4.0
Chipset	Qualcomm Snapdragon APQ8064 S4 Pro
Processor	Quad-core Krait; 384 - 1512 MHz; L0: 4 KB + 4 KB, L1: 16 KB + 16 KB, L2: 2 MB;
RAM	2G ; Dual-channel 533 MHz LPDDR2
Display	4.7 in diagonal IPS; 1280768 px; 320 dpi
GPU	Adreno 320
Radio	Integrated 3G/4G World/multimode
Wi-Fi	Integrated digital core 802.11n (2.4/5GHz)

Table 3.2: The specification of Google Nexus 4.

to install applications and modify the system configurations.

3.3.2 Application Power Stability

To build an accurate battery time prediction model, one of the assumptions we make is that the application power is relatively stable. We classify users into different types according to their application usage which leads to various user power behavior. Generally speaking, to finish the same task, the application power should be the same. We run several applications to do the repeated tasks and monitor the device power variation. Figure 3.2 presents the device power variation when we played with *Gallery* application. After viewing two photos, we modified them by adding filters, cropping it, rotating it and saving it. As the figure shows, there are power peaks at the beginning of viewing. They are usually caused by the user interaction like touch, click etc. and display rending. The power trend in the two viewing and editing are in similar manners respectively and the differences of average power are 18 mW and 32 mW. Compared with the 900 mW viewing power and 1850 mW editing power, the *Gallery* tasks' power are relatively stable. To further prove application power stability, we randomly chose applications in each category and repeated the experiments. Playing games is one of the main uses of mobile devices, and it consumes a large amount of energy. The power behavior of *TempleRun2* is described in Figure 3.3. When the application is loading, the power stays around 1800 mW for several seconds then decreases. Although the real time power fluctuated, the average power is around 1400 mW. Figure 3.4 illustrates the power information when an-

Category	Application	Description
Photography	Gallery	View and edit photo.
Business	Amazon	Go over the popular items, choose several items to see the detailed info.
	Dealmoon	
News	BBC	Go over the popular news.
	SinaNews	
Travel	Yelp	Search near restaurants, see the photo.
Games	CandyCrush	Complete the first two levels.
	Temple Run	
	AngryBirds	
Phone	Phone Call	Make and answer phone call.
Media	YouTube	Search a MV and play several mins.
	Youku	
Music	Pandora	Random choose a channel, listen for several mins.
	Douban	
Email	Gmail	Read 10 latest emails, send one email out.
	Default Email	
Weather	WeatherChannel	Search two city weather, check detailed info.
	YahooWeather	
Social Networking	QQ	Communicate through instant massage.
	Facebook	Go over several new states, then post news.
	Twitter	
Navigation	Map	Search road info from local to mall.
Utility	Calculator	Multiply and divide random number.

Table 3.3: Applications used in the analysis.

swering a phone call, which is the very basic function of a mobile phone. The average power of ringing and communication are 1690 mW and 1270 mW respectively. The communication real time power is more stable than the ringing part. The power difference of the two phone calls is less than 15 mW.

As the results show, the power behavior of applications is relatively stable when they perform the same tasks. Thus, it is reasonable to predict the battery time according to the user application usage. For the same user, the applications' power are stable since the usage scenarios are the same. The user evolution problem, for example, some users may like viewing photos first several months while later they may prefer to edit their photos, is out of the scope.

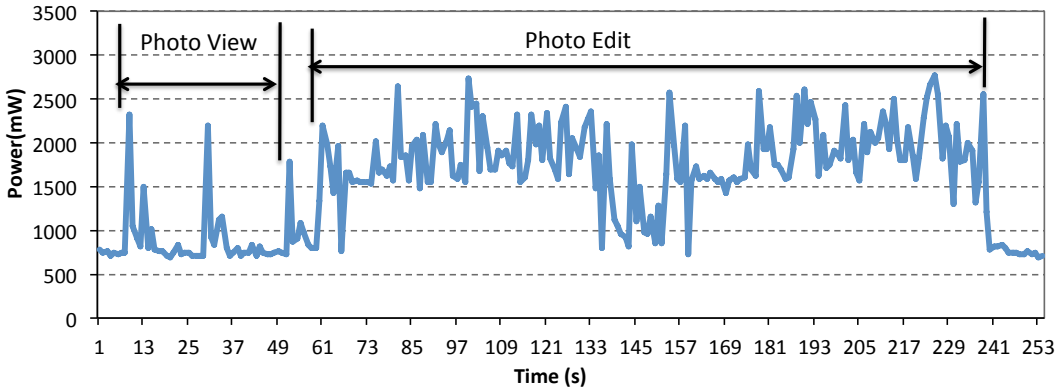


Figure 3.2: The device power variation of Gallery.

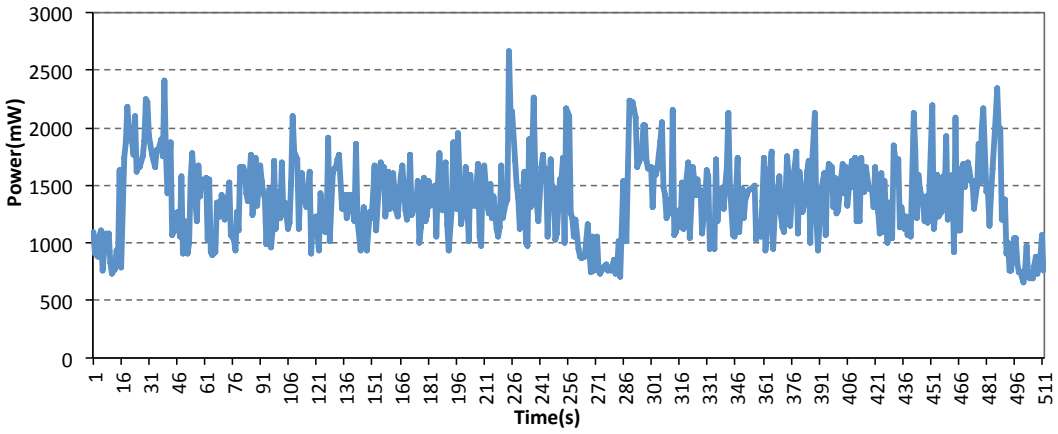


Figure 3.3: The power behavior of playing TempleRun2.

3.3.3 Application Power Accuracy Analysis

After presenting the application power is relatively stable, the next step is how to conveniently get the power. Similar with previous work [55], we leverage power models which are based on the resource usage information to calculate applications' power. From the prediction model, we can see that the accuracy of application power directly influences the prediction accuracy. We evaluated the power models by comparing the estimated power with the measured power. The experiments are done on the popular applications listed in Table 4.3. The power error is defined as Equation 3.3, which is different for each category. Figure 3.5 illustrates the comparison of the estimated power and measured power of *Pandora*. The total experiment time is about nine minutes, we paused the music after playing it for a while and then resumed

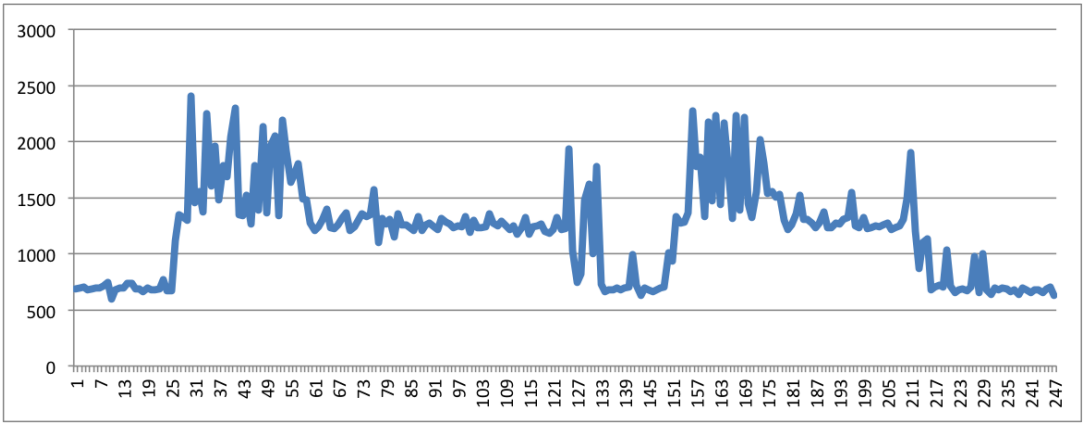


Figure 3.4: The power variation of answering a phone call.

it. The red line represents the estimated power, which is higher than real power when playing the music. When *Pandora* was paused, the measured power and the estimated power were the same.The error of the whole process is as high as 11.73%. For social applications, the estimated power is much more accurate. The power comparison for *Facebook* is described in Figure 3.6. We first viewed the news of friends and then posted a status without photos. The power trend for estimated and measured are similar while the estimated power is a little bit lower than measured power. The *Facebook* power error is about 4.72%.

$$error = \frac{|estimated - measured|}{measured} \tag{3.3}$$

Since the diversity in applications usage [25], we took the average error of all application

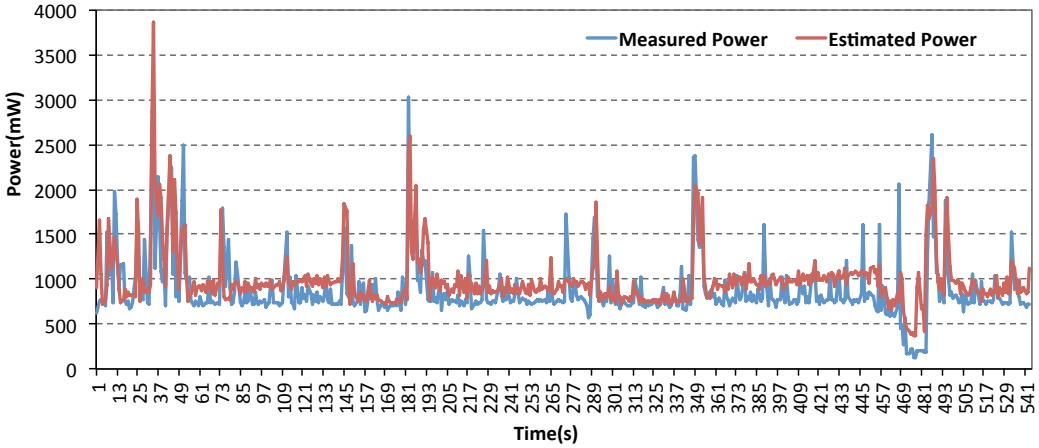


Figure 3.5: The power comparison for Pandora.

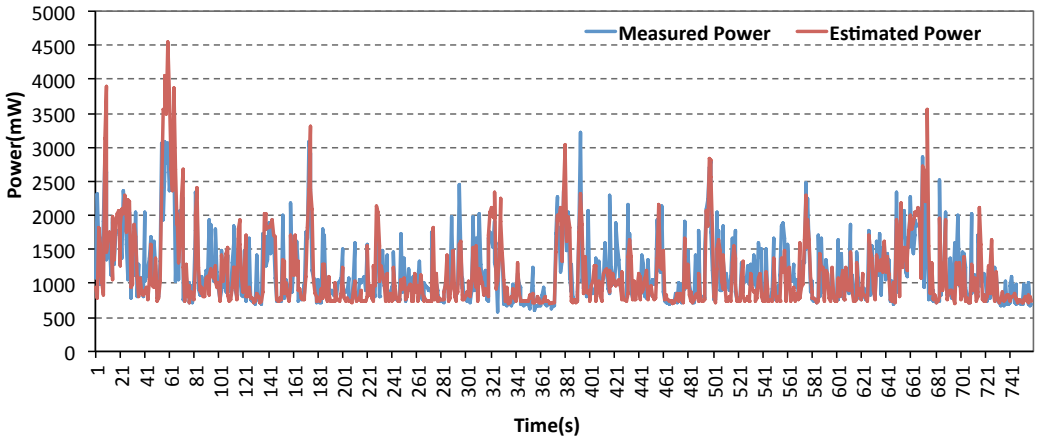


Figure 3.6: The power comparison for Facebook.

categories as the power models' error. Figure 3.7 shows the error information for the part of the popular applications. The red part is the estimated power subtracts the measured power, so it may be negative for some applications. For *Media* and *Games*, the estimated power is usually greater than measured power and the error is around 10%. For *Social* and *Business* applications, for example, *Amazon*, the error is much smaller and it is about 2%. As a result, the average application power error of the power models is 7.31%.

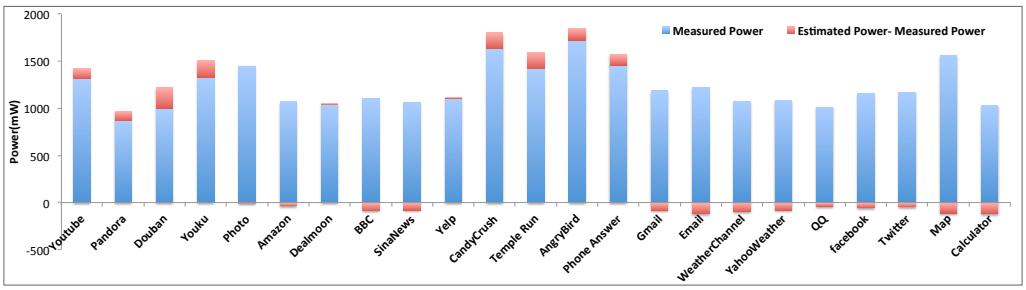


Figure 3.7: The distribution of estimated power error for popular applications.

3.3.4 Error of the Lifetime Prediction Model

After analyzing the application power accuracy, we illustrated its influence on the battery lifetime prediction model in this section.

Suppose the error of a_i (average application power) is e, which means

$$\forall i = 1,2,...,n. \quad |a_{iReal} - a_{iEstimate}| < e * |a_{iReal}|$$

Then, because

$$\sum_{i=1}^{n} b_i = 1$$

, and

$$\forall i = 1,2,...,n. \quad 0 \leq b_i \leq 1$$

So

$$|\vec{A_{real}} * \vec{B}^T - \vec{A_{estimate}} * \vec{B}^T| < e * |\vec{A_{real}} * \vec{B}^T|$$

Then the error of estimated life time is:

$$|\frac{t_{estimate} - t_{real}}{t_{real}}| = \frac{\frac{E}{\vec{A_{estimate}} * \vec{B}^T} - \frac{E}{\vec{A_{real}} * \vec{B}^T}}{\frac{E}{\vec{A_{real}} * \vec{B}^T}}$$

$$= \frac{|\vec{A_{real}} * \vec{B}^T - \vec{A_{estimate}} * \vec{B}^T|}{\vec{A_{estimate}} * \vec{B}^T} < \frac{e * \vec{A_{real}} * \vec{B}^T}{\vec{A_{estimate}} * \vec{B}^T}$$

$$< \frac{e}{1-e}$$

Since we can measure the current lifetime, and

$$r = \frac{t_{improved}}{t_{original}}$$

so the error of r is also $\frac{e}{1-e}$.

Figure 3.8 illustrates the trend of the battery life prediction error. When the application power estimation error is within 10%, the battery life prediction error is almost the same as the estimation error. When it is greater than 20% (battery prediciton error 25%), the prediction error is much larger than power estimation error. Generally speaking, the error of battery prediction is less than 10% since we chose a more accurate application power estimation approach.

3.3.5 User Behavior Analysis

User behavior is one of the important factors that affect battery lifetime. There are two datasets used in the dissertation to analyze user behavior. Dataset1 is the subset of the LiveLab

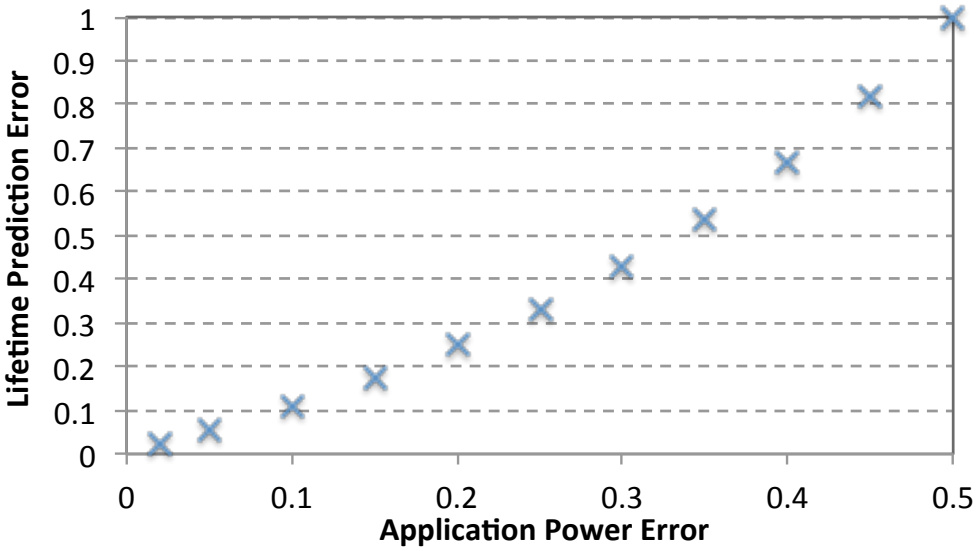


Figure 3.8: The relationship of battery prediction error and application power estimation error.

trace [98], which contains 34 iPhone 3GS users' application usage, phone call and sleep data from October 2010 to January 2011. Dataset2 was collected by ourselves, it has 14 students' usage information for one week. Table 3.4 summarizes the two datasets.

	#Users	**Platform**	**Duration**	**#Apps**	**#Categories**
Dateset1	34	iPhone 3GS	12 weeks	2400	26
Dateset2	14	Google Nexus 4	1 week	135	21

Table 3.4: An overview of the two datasets.

Dataset1 was used as training data and we classified dataset2 users into different types according to their application usage. *Phone call*, *SMS* and *Sleep* were treated as normal applications for consistency. Since there are 2400 applications, we preferred to choose application categories as the cluster metrics. To identify natural groupings of the user behavior data, we applied Fuzzy C-Means (FCM) clustering algorithm [99], which gives best result for overlapped dataset and comparatively better than k-means algorithm. The centroid of a cluster is the mean of all points in the dataset weighted by their degree of belonging to the cluster. The weight is updated every iteration as Equation 3.4 shows, and the distance we used was Euclidean distance (Equation 3.5). The level of cluster fuzziness m was set to 2. All the data are normalized

before clustering.

$$weight_k(x) = \frac{1}{\sum_{j=1}^{ClusterSize} \left(\frac{d(center_k,x)}{d(center_j,x)} \right)^{\frac{2}{m-1}}} \tag{3.4}$$

$$distance(k,x) = \left(\sum_{i=1}^{FeatureSize} (k_i - x_i)^2 \right)^{\frac{1}{2}} \tag{3.5}$$

Figure 3.9 illustrates the within error of cluster under different cluster sizes. The classification becomes more accurate as the size of cluster increase. We chose the size 6 because the improvement after 6 clusters is not as much as before. The data distribution among clusters is presented in Figure 3.10 and corresponding application usage information is demonstrated in Table 3.5. We listed major application categories in the table, each data is the average percentage of the running time of all applications in the category over the total logging time per day. Most time the devices were in the sleep state, except for user type T_2 which usage was dominated by phone call. Cluster 2 only has 1.7% items, its data is generated mainly by one user who made phone calls a lot and barely played with other applications. Asides from *Sleep*, users spent more time on social networks, browsers, games, media and photograph applications. After applying the cluster information to dataset 2, the result show it covers four user types and each user belongs to two types on average during the one week. To further explore user behavior information, Figure 3.11 illustrates the user type variation of one randomly picked user during 12 weeks. Hence, the user behavior in application usage has patterns that can be predicted but it is not simply fixed. Most of the time the user belonged to T_3, but he also changed to T_1, T_2, T_5, T_6 sometimes. We can use recent history to predict or take a weighted average value as usage information. Figuring out an accurate user behavior prediction mechanism is not the focus of the dissertation. For simplicity, we assume user behavior information is known.

3.3.6 Prediction Model Verification

To verify the prediction model can accurately calculate the battery lifetime, we simulated the normal usage cases under a controllable environment and compared the estimated battery

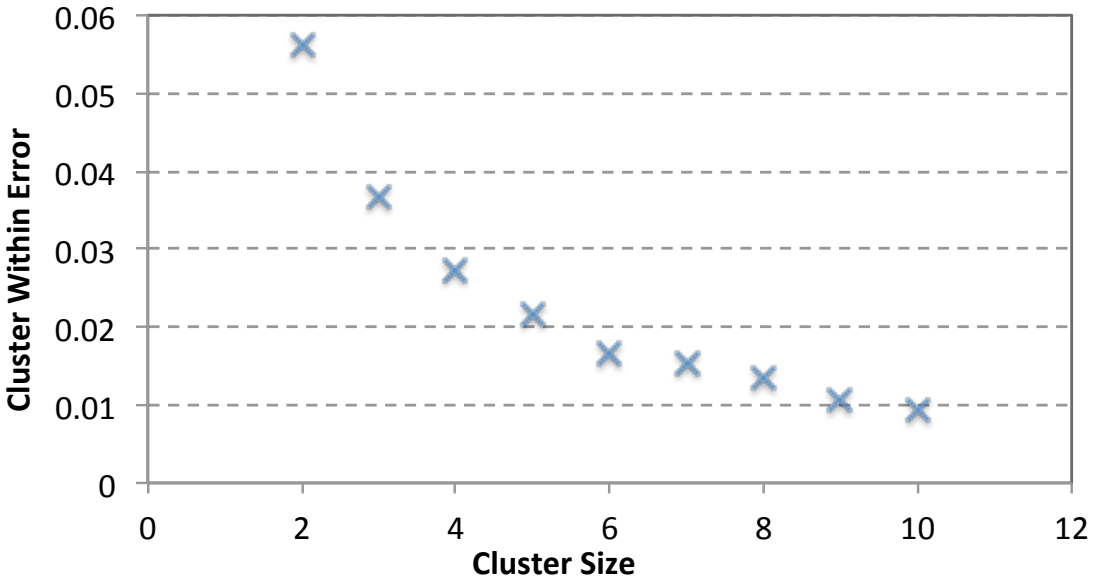


Figure 3.9: The cluster within error for different cluster size.

time with the real discharging time. In the experiments, all the parameters were logged twice per second so that the real time application power can be calculated. The running time of each application was predefined, and the user behavior monitoring tool also ran in the background to make sure the schedule was followed. The detailed experiment information is illustrated in Table 3.7. As we mentioned above, the application power is relatively stable while different tasks consume different power. We took the average power as the application power, for example, we did not distinguish video playing power and video searching power for *YouTube*. According to the logged information, the average power of applications and their running time were calculated. Multiplying the applications power and their running time percentage, the result 1206.1 mW was the average power in this experiment. The battery capacity was also recorded with the same frequency, it decreased from 72% to 40% and the full capacity is 2097 mAh. The battery voltage is 3.8V, so the energy consumed was 2549.95 mWh. According to the prediction model, this amount of energy should support 126.85 minutes. Compared with 118 minutes, the error of the prediction model is 7.5%. In reality, we cannot log the parameters in the background all the time since the monitoring program itself is energy consuming. At least,

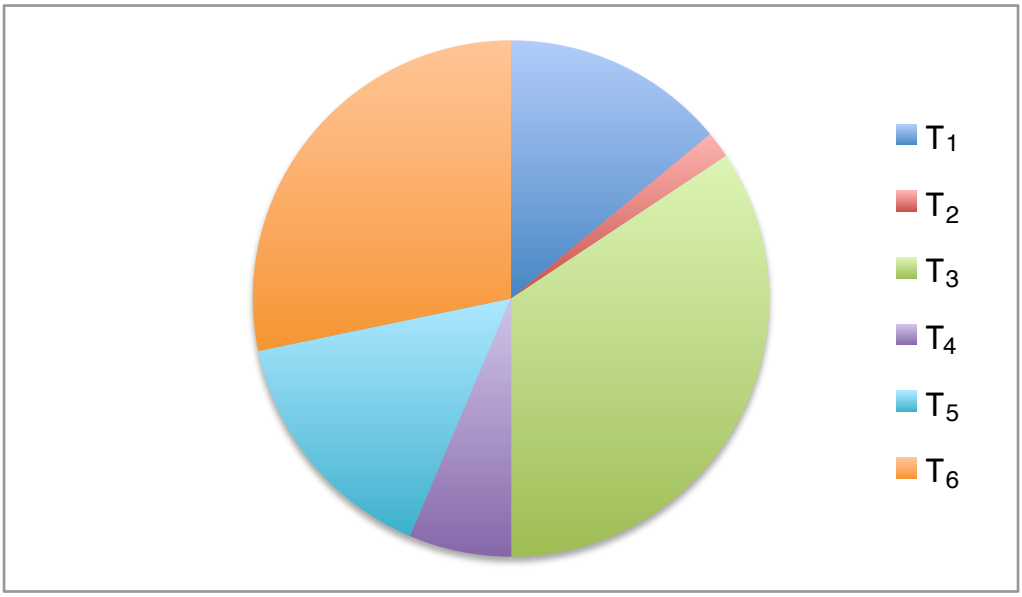


Figure 3.10: The data distribution for six user types.

there is no such low overhead service that provides application real time power information. So another option is leveraging history average applications' power rather than calculating power from real time parameters. The result shows the error was less than 1% in the same experiment.

The prediction model error is less than we calculated since the battery capacity is not very accurate. The voltage decreases as the battery capacity decreases [100], and in our calculation the voltage is 3.8V all the time. So the real battery capacity is less than 2549.95 mWh. Besides, the granularity of battery capacity is 1%. There was at most 209 mAh that we did not know if it was used or still reserved, which depends on the battery capacity update policy. Moreover, part of the error is caused by the power models that are used to estimate application power as Section 3.3.4 demonstrated. Hence, the prediction model is reasonably accurate to calculate the potential battery lifetime.

3.4 Applications of Prediction Model

After validating the battery life prediction model, we analyzed the potential battery extended time for different user types according to the model and the future trend with the improvement of hardware components.

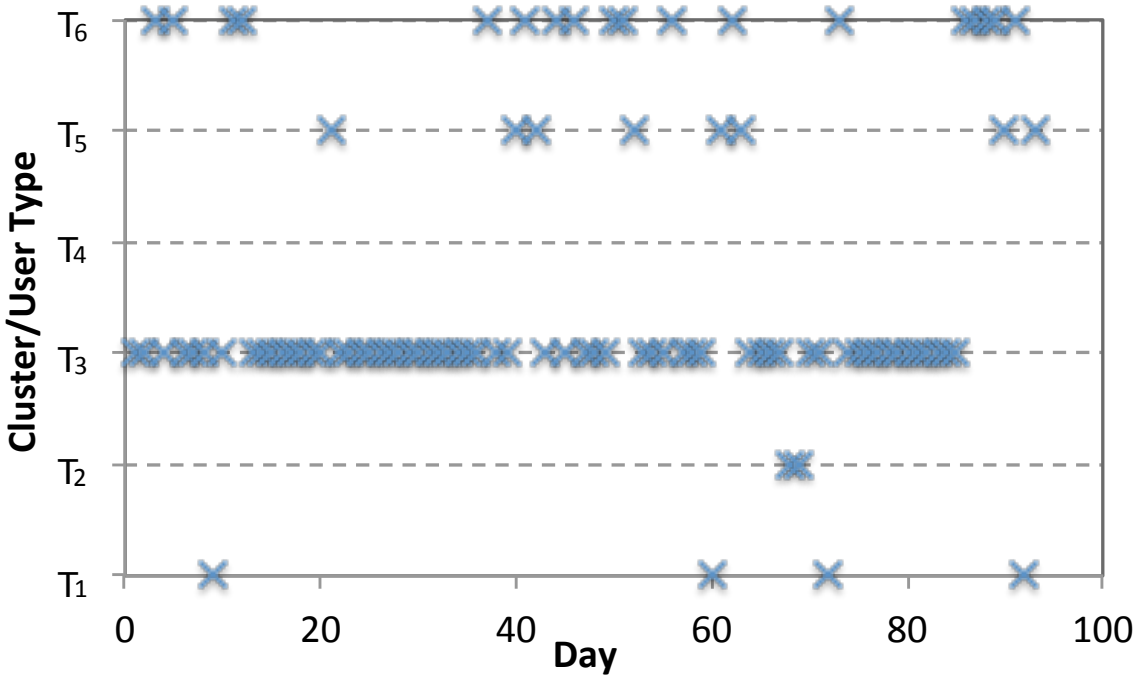


Figure 3.11: User cluster type changes with time.

3.4.1 The Theoretical Battery Lifetime

For the same device, battery lifetime for different user are also different. We presented the theoretical battery lifetime for the six user types based on the typical application usage information and calculated the corresponding potential battery extended ratio by changing the user behavior.

Although the power of each application is different whether they are in the same category or not, it is impossible to gather every application's power and analyze the battery time for every combination. For our experiment, we classified users according to the time they spend on each application category. We used the average power of applications we tested to represent "category power", such as 1591 mW for *Games*, 1081 mW for *News*. In the specification for Google Nexus 4 [101], the battery capacity is 2100 mAh, the standby time, talk time, video playback and web browsing time are 250 h, 10 h, 8 h and 7 h respectively. Hence, we can calculate the corresponding power. For all other applications which do not belong to categories listed in Table 3.5, we use 1000 mAh to represent their power. The summary of each category

power is presented in the first row of Table 3.6. Hence, we have the value of $\vec{A}$ and $\vec{B}$ illustrated in prediction model in Section 3.2. The second row of Table 3.6 is battery discharging time if the system only has the corresponding applications running. It gives the intuitive impression of the category power. The time varies from 5 hours to 10 hours except *Sleep*.

Since making phone calls and sending messages are the basic functions for a phone, we did not modify their usage information in the following analysis. For the T_1 type of users, the estimated battery time was 17.56h based on the application usage. The users spent nearly the same time on *Browser* and *Social Networking* applications, following that was the *Media* applications which occupy 3.89% of total time. If we only keep these top three categories as well as *Phone Call* and *SMS* and count other applications time on *Sleep*, the battery can last 29.6h. For users in type T_2, they treated the device as a basic phone and spent over 95% time on *Phone Call*. So there is not much time that can be extended for this kind of user. After allocation the rest 2.91% time to *Sleep*, the battery time increased to 10.27h which is 1.9% longer than before. Compared with over 50% increase for user type T_1, it is too small to make a difference. Similar with user type T_2, user type T_3 also contains a category, *Sleep*, that is responsible for over 90% time. The battery time for the typical T_3 type of users is 66.28h since most of the time the device is in sleep state. If the user does not play any applications except *Phone Call* and *SMS*, the battery time will be extended to 147.3h. It is because the *Sleep* power is much lower than other applications and the *Sleep* power also dominated the energy consumption in the original case , the battery can last much longer by increasing 5.73% time for *Sleep*.

For user type T_4, the top categories were *Social Networking*, *Browser* and *Media*. The time spent on *Social Networking* is 7.89%, while the other two occupied almost the same amount of time which is around 4.5%. After maintaining these three categories, the estimated battery life time was 23.43h. Compared with the former 10.45h, it doubled the battery lifetime. Similarly, users in type T_5 also preferred the three categories, while the percent is around 3.42%. The

potential battery life was 13h longer than the typical case. At last, for user type T_6, there is one more category that should be considered asides from the top three since *Photography* and *Media* have the same percentage. The battery life can be increased up to 40%.

The summary of the potential extended battery life for each user type is also presented in Table 3.6. The r is the increased battery time over the corresponding original time as defined in Equation 3.2. Based on the results showed in the table, we conclude several implications as follows:

(1) The battery life for users who prefer one specific application category is difficult to increase. For example, users in type T_2 like *Phone Call* and the ratio r is only 1.03. We can not sacrifice the most preferred applications while others only have small influence on the device power.

(2) The *Sleep* time decides the battery time since the power in sleep state is too small compared with other applications. For T_4 users, the *Sleep* time increased by 40% and the battery life doubled. The ratio was over 1.6 in user type T_1 and T_5 as the corresponding time in sleep state increased to over 10%.

(3) The battery life can be extended up to 40% if users adjust application usage rather than put the device into sleep state. The difference of category power usually is around 500 mW. In some extreme cases, the difference can be 1000 mW. The time spent on each application category is less than 10% in most cases. Hence, if a user gives up a high power application and puts the saved time on another low power application, the difference of average power is about 100 mW. If the device average power decreases from 350 mW to 250 mW, the battery lifetime is 40% longer. With the increase of the original average power, the potential extended battery life is decreased, 10% for 1050 mW.

For mobile devices energy saving, the proposed strategies need to be more personalized. Users can set proper system and application configurations to extended battery life. For example, the pull option for email synchronization is more energy efficient than push for users who receive a lot of emails everyday. Besides, same energy optimization approaches will have

different influence for different users. Users who like playing games will enjoy the energy savings on the GPU and CPU, while for the users who usually use the device to listen to the music probably will not notice the battery lifetime increased.

3.4.2 Hardware Component Improvement

As more and more researchers work on mobile devices energy saving field, a lot of optimization approaches are proposed for the system and various hardware components. We discussed the influence of the five main component improvements, which includes battery, CPU, radio, display and Wi-Fi, on the battery lifetime for each user type .

The hardware improvements correspond to the parameter s_i (and thus $\vec{M'} and \vec{A'}$) illustrated in the prediction model in Section 3.2. For example, if the energy efficiency of CPU improves 10 times then s_1 is 1/10, and if the energy efficiency of radio improves 2 times, then s_3 is 1/2. Hence, the primary inputs are the power consumptions of each component for every kind of applications and the improved energy efficiency of hardware components. We can use the Equation 3.2 to obtain the extended ratio of battery lifetime for each user type.

We logged the power consumptions of each component in previous experiments in Section 3.3. For component improvement, the increased times of energy efficiency are demonstrated in Table 3.8. According to the Moore's law, the performance should be doubled in 18 months. So the improvement should be 2^3, nearly 10 times, in the next five years. However, the components are not all energy proportional. *CPU*, as the most important component in the mobile devices, is used by all applications, its energy efficiency may increase up to 10 times. Battery capacity is hard to increase as its developing history suggests, but with the emerging of the wearable devices, the industry is stimulated to develop new technology to satisfy the demand, which may help the battery increase up to 10 times in the next five years. Similar situations for display, asides from the big improvement (10 times), the energy efficiency may also increase a little because its performance is acceptable and the appearance of new technology is hard to predict. We also considered two potential improvement for Wi-Fi as it supports

the favorite function, wireless, of mobile devices. The characteristics of *radio* make it need to monitor phone calls and messages all the time and they are basic functions of a phone. We think its energy efficiency may be doubled in the future. Hence, we calculate the four cases listed in Table 3.8 that describe the trend of energy efficiency improvement for mobile device components in the next five years.

The rest of information in the Table 3.8 presents the results of how many times that the battery life can be extended for each user type in each of the four cases. The extended ratio is grows linearly to the battery improvement, the results are at least greater than 10. If the improvements of all the components are the same, the result is also a linear function which is not related to user behavior, but this situation is very rare. We can see that the results for case 1 and case 2 are around 30, while the results for case 3 and case 4 vary between 40 to 58 (except for users in T_3). The difference between the two groups is the display improvement. Because the average power of the display of all application categories is almost 40% of the device power, its influence can double the battery extended ratio at most. For users in type T_3 who put their device in the sleep state for most time, the impact is not as great as in other cases. In case 4, T_4 users can enjoy 58.7 times battery life time extension, which is about 2 times more than T_3 users. The results show that user behavior can affect the battery life time a lot even for the same hardware improvement.

Next, we look at how much the hardware should be improved to achieve a target battery life, such as seven days, for different types of users. The average power of the main components for each application category is known as shown above. We can calculate the user power demand for each component according to the application usage. Suppose we want to improve the battery life by r times, the capacity of the battery can improve x_0 times, and denote

$$N = (\vec{M'_1}; \vec{M'_2}; ...; \vec{M'_n})$$

, then we can calculate how much should be improved the by solving a linear equation

$$sum(N * \vec{B}^T) = \frac{x_0}{r}$$

Note that N is a matrix and since $\vec{M}_i'$ is just s_i times each elements in $\vec{M}_i$, so the variables in the linear equation are s_i. For example, assume the current battery life of users in type T_2 is about 6.29 hours, which is calculated by using the measured average power. The standard battery capacity for the Google Nexus 4 is 2100mAh (7980mWh). The current average power of the CPU, Radio, Wi-Fi and display for users in T_2 are 479.5 mW, 391.1 mW, 0.6 mW and 397.6 mW respectively. To extend it to seven days, the lifetime should be improved by 26.7 times. Intuitively, it can be achieved by increasing the battery capacity by 26.7 times. Assuming the capacity of the battery can improve by 10 times, then the energy efficiency of CPU, radio, and display should all improve 2.67 times. The improvement of components can vary greatly. For T_4 users to enjoy a one week experience, if we assume that the capacity of the battery can only improve 2 times, then we need to improve the energy efficiency of CPU by 7.6 times, display by 6 times, Wi-Fi by 1.45 times and radio by 3 times.

Generally speaking, the optimization approaches for components which are required by most applications have more impact, such as battery, CPU and display. For different users, the improvement of the "most used" components which are inferred from user behavior is more effective for them. Moreover, we discussed the theoretical influence of components' improvement, while the actual results should also consider the software interference.

3.5 Summary

In this chapter, we investigated the influence of user behavior on the energy consumption of smartphones. We built a prediction model estimates the battery lifetime based on user behavior and usage information of hardware components. In order to analyze the influence of user behavior, we classified users into six groups according to their application usage pattern and estimated the theoretical maximum battery life they can achieve. We presented the potential time that battery can be extended if the user behavior evolves as well as improving main hardware components. Besides, we also analyzed how much work needs to be done if we want to extend battery life to reach a certain goal from the two aspects.

In the next chapter, we will introduce how to estimate application and system power dissipation based on their hardware usage. The power profiling information is used to evaluate the energy efficiency of different system configurations and energy saving approaches.

User Type	Utilities	News and Magazines	Email	Games	Media	Photography	Browser	Social Networking	Weather	Phone Call	Sleep	SMS
T_1	1.08%	0.49%	1.18%	1.75%	3.89%	2.56%	4.08%	4.69%	0.58%	5.27%	59.01%	6.47%
T_2	0.12%	0.05%	0.08%	0.08%	0.25%	0.16%	0.21%	0.21%	0.09%	96.27%	0.60%	0.82%
T_3	0.34%	0.06%	0.34%	0.62%	0.60%	0.73%	0.89%	1.12%	0.30%	1.21%	91.70%	1.36%
T_4	2.29%	0.94%	1.49%	1.69%	4.39%	2.08%	4.83%	6.57%	0.78%	7.89%	26.85%	8.22%
T_5	0.93%	0.32%	0.89%	1.71%	3.09%	2.23%	3.65%	3.53%	0.55%	4.12%	68.82%	5.07%
T_6	0.68%	0.20%	0.67%	1.30%	1.74%	1.75%	2.32%	2.41%	0.45%	2.71%	79.21%	3.52%

Table 3.5: The application usage information for each user type.

[a]

[a]

	Utilities	News and Magazines	Email	Games	Media	Photography	Browser	Social Networking	Weather	Phone Call	Sleep	SMS	r
Power (mW)	1032	1081	1201	1591	997	1407	1140	1163	1057	798	32	981	
Time (h)	7.73	7.38	6.65	5	8	5.67	7	6.86	7.55	10	249.37	8.13	
T_1					✓		✓	✓		✓	✓	✓	1.68
T_2										✓	✓	✓	1.03
T_3										✓	✓	✓	2.22
T_4					✓		✓	✓		✓	✓	✓	2.24
T_5					✓		✓	✓		✓	✓	✓	1.61
T_6					✓	✓	✓	✓		✓	✓	✓	1.39

Table 3.6: The category power and the summary of potential battery extended for each user type (T_1 to T_6).

a

Application	**Estimated Average Power(mW)**	**History Average Power (mW)**	**Duration (mins)**	**Percentage**
Temple Run 2	1541.69	1593.86	14	11.87%
Dealmoon	1281.06	1053.33	26	22.03%
YouTube	1233.44	1419.68	48	40.68%
Weather	924.88	974.47	2	1.69%
Twitter	1147.97	1125.48	4	3.39%
Photo	867.00	1440.71	9	7.63%
BBC News	875.65	1012.94	15	12.71%

Table 3.7: The detailed power and time information for applications tested in the experiment.

	Battery	**CPU**	**Radio**	**Wi-Fi**	**Display**	T_1	T_2	T_3	T_4	T_5	T_6
Case 1	10	10	2	2	2	29.77	28.66	21.32	31.29	28.81	26.75
Case 2	10	10	2	10	2	30.42	28.67	21.59	32.02	29.42	27.25
Case 3	10	10	2	2	10	50.24	44.73	27.68	56.32	46.92	40.56
Case 4	10	10	2	10	10	52.12	44.76	28.13	58.73	48.55	41.71

Table 3.8: Life time improvement for different users and cases. (T_1 to T_6 are user types.)

CHAPTER 4 POWER PROFILING ON MOBILE DEVICES

To deal with the battery drain problem, we first need to understand the power/energy behavior of applications and systems. We designed and implemented **Bugu** which aims to analyze power and event information and providing users with detailed energy behavior data. We analyzed popular applications' power behavior on different platforms. The results showed several interesting observations that indicate the potential energy optimization for both applications and systems. We further revealed the underlying reason of different power consumption for several applications.

In the following sections of this chapter, we give brief introduction in Section 4.1. The Bugu service and the implementation of its components are presented in Section4.2 and Section 4.3 respectively. Then, we demonstrate our experiments results in Section 4.4 and propose four critical implications for energy-efficient mobile application and system design in Section 4.5.

4.1 Introduction

Nowadays, mobile devices, such as tablets and smart phones, have become an important part of our daily life. According to a statistical report of Cisco[102], by the end of 2014, the number of mobile-connected devices will exceed the number of people on earth, and there will be nearly 1.4 mobile devices per person in the near future. At the same time, the development of mobile devices also stimulates the application market. The number of Android applications increased 50% in last year, which is over 1,200,000 [2].

There is no doubt that these applications make our life more convenient and colorful, but they are also big energy consumers on mobile devices and significantly influence battery lifetime and user experience [103]. As an end user, we want to know *"For the same functionality, which application is more energy-friendly?"* Except the battery issue, energy efficient applications are more competitive on the market. In a green software awareness survey [104], data shows about 70% people believe that optimizing software is an effective way to save energy and 58% of respondents would select software applications which have energy level labels on them. Application developers often ask the question: *"Why do my applications consume such*

amount of power?" especially for mobile devices. System developers focus on the whole system, not just some components or specific applications. Answering the question *"How to save and effectively control system power?"* is the final goal of system developers. However, the first step to answer these questions is to understand the energy consumption of the system and applications.

We design and implement the Bugu service, which is an application level power profiler and analyzer. As Figure 4.1 illustrates, the Bugu server returns related applications' power information to end users and gives them more suggestions when they choose applications. For application developers, aside from the similar applications' power information gathered from the server, the Bugu client also shows the event information of their applications, so that application power problems can be easily distinguished. From the viewpoint of system developers, detailed system power information provided by the Bugu client is helpful for them to adopt power saving mechanisms. Leveraging Bugu, we analyzed 100 popular applications and revealed the root causes of high power consumption for some of them in case studies. From the result we observed several aspects that can be improved to save both application and system energy, such as sensors and video module energy efficiency, phone service *rild*.

4.2 System Design

The Bugu service is mainly designed for system designers, mobile application developers and end users. Thus, the Bugu service not only presents the application-level power consumption on a single device, but also supplies a group of REST (Representational State Transfer) style APIs for users to share and compare power data. It also supplies event information that may help them to understand the underlying reasons that cause the power consumption.

As Figure 4.1 describes, the Bugu service includes two parts: the Bugu server and the Bugu client. The Bugu server collects applications' power information on each device and supports the Bugu client with these data. The Bugu client is used to monitor application power consumption, monitor events and analyze these information. The results are presented in tables

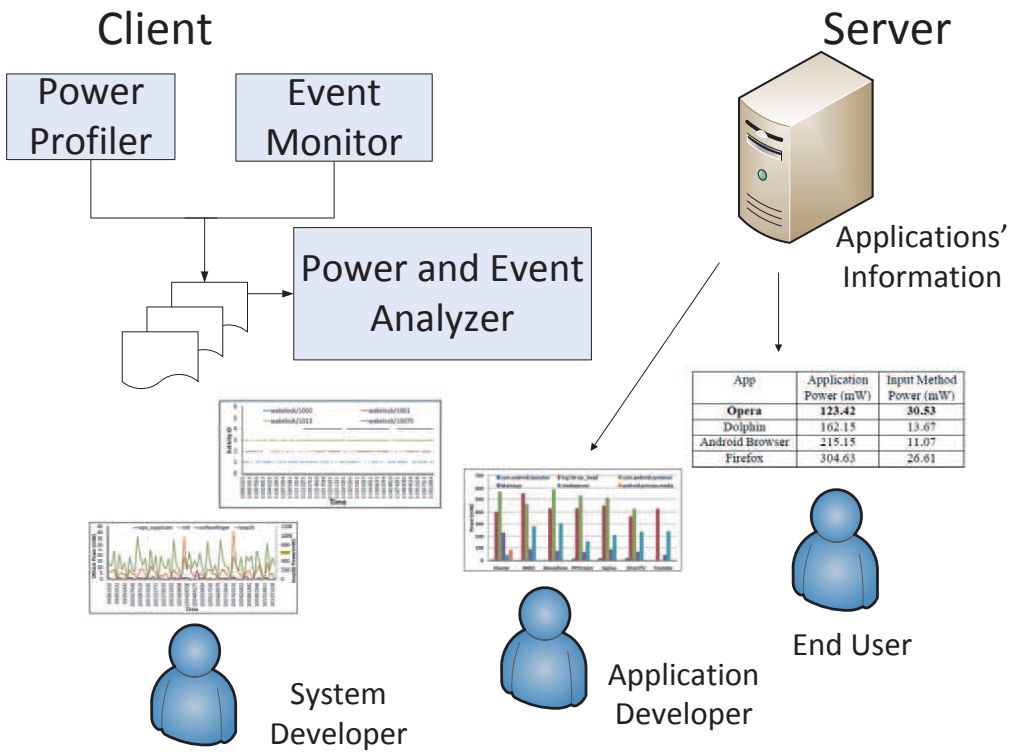


Figure 4.1: The overview of Bugu.

Table 4.1: The energy models.

Components	Energy Models
CPU	$E_{CPU} = \sum_{i=1}^{NumberOfSteps} Time_i * (IdlePower + MaxPower * U)$ $U = (\Delta T_{sys} + \Delta T_{user})/(\Delta T * CoreNumber)$
Wi-Fi	$E_{wifi} = WifiOnAvgPower * WifiOnTime + WifiActiveAvgPower * WifiActiveTime$
Screen	$E_{Screen} = \sum_{i=1}^{NumOfBrightness} (Time_i * (i/NumOfBrightness * ScreenFullPower))$
Bluetooth	$E_{bluetooth} = BtOnAvgPower * BtOnTime + BtAvgPowerAtCMD * BtPingTime$
Radio	$E_{radio} = \sum_{i=1}^{NumOfSignalBin} (SignalTime_i * SignalAvgPower_i) + RadioScanAvgPower$ $*RadioScanTime + PhoneOnTime * RadioActiveAvgPower$

and figures for easy understanding and comparison.

4.2.1 The Bugu Server

The Bugu server has two functions: collecting application power information from the Bugu client and supplying these power information to users. Users can contribute their data to

the Bugu server by uploading their profiling records, which will help future customers. With the first function, we maintain a large database of application power consumption information on different types of mobile devices. After ranking these applications, users can get better understanding before installing them. The building of this database requires users' contribution so we can cover as much applications and devices as possible. It is a huge and continuous work. At present, we provide power information for most popular applications on several Android devices that we have.

Based on the type of device and the type of application the user wants to compare, the Bugu server finds the related power information and delivers it to the Bugu client. Then, end users know the comparison results of these applications, they can choose an energy-friendly one to install. For application developers, they can compare the power consumption with the application they developed to evaluate their products.

4.2.2 The Bugu Client

The Bugu client has three main functions: estimating application-level power consumption, monitoring system and application events and displaying the information to the user in a meaningful way. It is composed by power profiler, event monitor, power and event analyzer and user interface module. The procedure is as follows: the power profiler and event monitor record the raw data they need; and the analyzer extracts the data and sends application level power and event information to UI module which displays the data including other applications' information obtained from the Bugu server to users in a meaningful way.

Power Profiler: Power profiler is responsible for estimating the system and application power consumption. It uses a group of energy models, which are listed in Table 4.1, to estimate energy consumption based on how much of each hardware resource was utilized by each application. With the time information, we calculate the average power consumption. The power profiler considers the following components: CPU, Wi-Fi, 3G, GPS, sensors, bluetooth, screen, radio, and so on. We leverage some energy models from our former paper [105], and tune the

parameters for mobile platform. For components like sensors, we build the energy models according to their different power states. Aside from application's power, we also record the power of hardware components in the system. So far, we do not consider screen power for each application, while it is available on system level. The reasons are as follows: from research of Dong *et al.* [106], we know that for OLED screen, different color presented can affect screen power. While the applications' user interface is part of their design style, it will affect user experience if the color is changed. For LCD display, the screen power is determined by brightness level, applications themselves can not save much on screen part. The optimization approaches we want to find are from a functionality aspect, not appearance. Although we can get application level screen power according to the time that an application is in foreground and pixel information, it is not very suitable and it increases overhead of Bugu.

The power profiler saves the power information in a formatted log file, which includes the utilization information of all the active applications on each component. The data is recorded once per second. With this information, system designers could analyze the underlying reasons that cause the energy consumption.

Event Monitor Aside from just monitoring the power consumption, Bugu can also monitor the events of system and applications. Those events include: wakelock, Wi-Fi state change, bluetooth state change, audio and video state change and different sensors on/off state. For example, we could know when an application acquired or released a wakelock. In the Android operating system, there are six types of wakelock, which represent the privilege to use several hardware devices. All of them make the processor keep in active state. Many applications drain battery so quickly because of misusing the wakelocks. Thus, those events are helpful for us to understand how the applications cause the power consumption.

For each event, we log the time, type and related information, such as the level of brightness, sensor states. The system developers use the information to deeply analyze the system power consumption problem.

Power and Event Analyzer The power and event analyzer is used to process the result recorded by the power profiler and the event monitor. The most important function of this module is to calculate the average power of each application. We write the algorithm to calculate application average power. First, we need to filter the effective data by detecting the longest active period of each application. We define application inactive state as its energy consumption does not change in N successive calculation points. The interval between two calculation points is one second. According to our experiments log, most of the applications are paused or went to background if we can not detect their energy variation after three times. Sometimes, energy kept the same because of the sampling delay. Hence, in our experiment, N equals three. Then based on the time period and the logged energy information with usage data to calculate the average power. The analyzing process is done off-line in order to lower the overhead of the Bugu client.

One of the challenges we faced is that the power monitor cannot accurately run periodically, that's because Android is not a realtime operating system. Thus, we improve our algorithm that when we compute the power during two time intervals, the record will be skipped if the time interval is smaller than the threshold. Otherwise, we may get abnormal power results because there is a delay before we obtain the utilization information.

4.3 Implementation

To implement the Bugu service, we not only developed the server program and the Android client application, but also modified and compiled the Android system to monitor the events. In this section, we describe how we implement the Bugu server, the power profiler and the event monitor

4.3.1 The Bugu Server

The Bugu server maintains application power information, gathers the information from users and provides comparison results to end users and application developers. When users send a request, the Bugu server returns the same category applications list and each item de-

```
<?xml version="1.0" encoding="utf-8"?>
<device name="Android">
                . . . . . .
    <item name="screen.on">49</item>
    <item name="bluetooth.active">142</item>
    <item name="bluetooth.on">0.3</item>
    <item name="dsp.video">88</item>
    <item name="radio.active">185</item>
    <item name="gps.on">50</item>
    <item name="cpu.idle">1.4</item>
    <item name="cpu.awake">44</item>
    <array name="cpu.active">
        <value>55.4</value>
        <value>82.1</value>
        <value>113.7</value>
        <value>205.4</value>
        <value>259.0</value>
    </array>
</device>
```

Figure 4.2: The resource file in Android system.

scribes the application name and its power consumption. There are two ways for users to contribute their data to the Bugu server. They can choose the *upload* option on their records, or write results on the submission page. We use REST [107] to implement our server, the request URI describes the parameters of the type of device, the type of application and the limit of returned results. The server interprets the request and send back the corresponding results.

4.3.2 Power Profiler

The power profiler is implemented as a service running in the background periodically. It requires the base power of hardware components and their utilization for each application to estimate the power. We get the base power information from the `PowerProfile` class of Android, which reads power values from a resource file (as Figure 4.2 presents). For example, we could get the power of the CPU when it is working on each power step, and the value under *cpu.active* corresponding to power consumption of different CPU frequencies. For the components that are not reachable from the `PowerProfile` class, we did some experiments that described in Section 4.4 to get their base power. In addition, we get most the application level resource utilization from the `BatteryStats` class. For each application running in the system,

their statistic information can be achieved from *batteryStats.getUidStats()*. Then the components utilization information is obtained by calling corresponding method: *getSensorStats(), getProcessStats(), getWakelockStats()* and so on. The audio and video time are achieved by modifying Android source code since the logging part have not implemented and the original results in `BatteryStats` are all 0. Some data are read from Linux file system, for example, the transmission packets for each process. All the results are already logged for each process, so it can be used directly in the real scenario. The List 4.1 presents the segment of code that describes how we use the information to estimate the energy consumption of each process.

```
private void processCPUPower(Uid.Proc ps)
{
    long userTime = ps.getUserTime(statsType);
    long systemTime = ps.getSystemTime(statsType);
    appPowerInfo.foregroundTime += ps.getForegroundTime(statsType) / 1000;
    appPowerInfo.cpuTime += (userTime + systemTime) * 10;
    int totalTimeAtSpeeds = 0;
    for (int step = 0; step < speedSteps; step++) {
        cpuSpeedStepTimes[step] = ps.getTimeAtCpuSpeedStep(step, statsType);//
            microseconds
        totalTimeAtSpeeds += cpuSpeedStepTimes[step];
    }

    if(totalTimeAtSpeeds > 0)
    {
        for (int step = 0; step < speedSteps; step++) {
            double ratio = (double) cpuSpeedStepTimes[step] * 1.0 /
                totalTimeAtSpeeds;
            appPowerInfo.cpuPower += ratio * appPowerInfo.cpuTime *
                speedStepAvgPower[step]; // milli joule * 1000
        }
    }
}
```

Listing 4.1: The example of CPU power calculation.

With the base power and utilization information, the power profiler computes the accumulated power consumption of each hardware component. The application power is the sum of

all components' power since the components' usage information is recorded for each process in `BatteryStats`. The detailed power and utilization data are logged for further analysis.

4.3.3 Event Monitor

The implementation of the event monitor requires Android system's support, so that we can monitor all the events information. We implement this function by modifying the `BatteryStatsService` class (as List 4.2 shows), which collects all the system and application events that related with battery usage. For each event noted in `BatteryStatsService`, we log its states and make it visible to users. When `BatteryStatsService` receives an event, it will broadcast an `Intent` message, which could be received and logged by the Bugu client.

```
public void noteStartWakelock(int uid, int pid, String name, int type) {
        enforceCallingPermission();
        synchronized (mStats) {
            mStats.noteStartWakeLocked(uid, pid, name, type);
        }
       if(enableEventListen){
            synchronized(helper)
            {
                helper.noteStartWakelock(uid, pid, name, type);
            }
        }
    }
```

Listing 4.2: The example of logging wakelock event.

In addition, we also use the event monitor to trigger energy-optimization actions by sending some special `Intent` messages to energy-aware services of the Android system. One message we sent is Wi-Fi tail, which will be generated when the Wi-Fi enters the tail stage. As far as the message is received, the system could leverage the tail stage to piggyback some asynchronous data, such as a post of twitter cached before.

4.4 Evaluation

In this section, we evaluate our work on tablet and smart phone. We present how the Bugu service works for three groups of users: end user, application developer and system developer.

After collecting 100 applications power data, we did some analysis and found several observations. Moreover, we analyze the overhead of Bugu and summarize the implications we got from the experiments.

4.4.1 Experiment Setup

As we described above, Bugu acquires system information from two classes, *PowerProfile* and *BatteryStats*, and calculate application power consumption based on our power models. To verify the accuracy of data read from *PowerProfile* class, we first compared the resource file between different Android OS versions. We found that the file is corresponding to mobile phone models, not the Android operating system. It is the same when we updated from Android 2.3 to Android 4.0. Besides, we wrote our own testing benchmarks to see if the results are consistent with the data recorded in the file. We mainly tested brightness, CPU, socket connection and file input/output. In our experiments, the benchmark applications run foreground and other applications were terminated to keep the accuracy. We connected a resistor between battery and phone, then attached the National Instruments devices [108] to record the voltage of the resistor and the phone. Hence, we got the current of the phone based on the resistor. After that we can calculate power as well as energy information. The resource file contains power data of different state of screen, wifi, cpu, bluetooth and so on. We run our benchmarks to compare the data we collected with the information in the file. Besides, we calculated file I/O data and put it in our power calculating system, which is not supplied in the resource file.

The mobile devices we used for these experiments are Google Nexus S and Motorola Xoom. Their hardware parameters are presented in Table 4.2 that includes sensors information. Both of these devices use Android 4.0.4 OS. According to AndroLib's statistics [109], there are 640,000 android applications in the market. To ensure representative results, the applications we choose cover most categories. They have over one million installs and ranked in Top-100 as claimed by Google Play. Table 4.3 lists the information and representative applications.

Hardware Components	Nexus S	Xoom
CPU	ARMv7 Processor rev 2	ARMv7 Processor rev 0
Frequency (MHz)	100 - 1000,5 steps	216 - 1000, 8 steps
RAM (MB)	335	718
Sensor	KR3DM Accelerometer GP2A Light Sensor AK8973 Magnetic Sensor	KXTF9 Accelerometer Ambient Light Sensor AK8975 Magnetic Sensor

Table 4.2: Experiment platforms.

Category	Applications
Business	Documents To Go, UPS Mobile, Pocket Cloud Remote, etc.
Game	Fruit Ninja, Temple Run 2, Talking Tom Cat, etc.
Finance	Google Finance, Expense Manager, TurboTax SnapTax, etc.
Health and Fitness	Instant Heart Rate, Workout Trainer, Lose It, etc.
Media and Video	YouTube, RealPlayer, Movies by Flixster, etc.
Music and Audio	iHeartRadio, Amazon MP3, Google Play Music, etc.
Education	Kids Animal Piano Free, How to Draw, Aldiko Book Reader, etc.
Tools	PicsArt, Barcode Scanner, Tiny Flashlight, etc.

Table 4.3: Summary of selected applications.

4.4.2 Bugu Case Studies

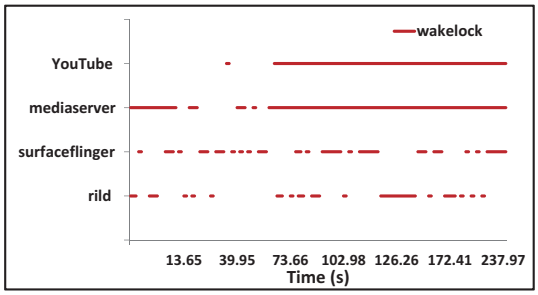
We introduce three case studies here to illustrate how different kinds of users can take advantage of the Bugu service.

End User Scenario An end user usually wants battery work longer without frequently charging. Aside from saving energy by operating system or shutting down unused devices, this goal can also be achieved through installing energy-friendly applications. The Bugu server maintains a lot of applications' power data with the hardware platform information. End users can request these information and search the category of the application they want to install, the data returned is ranked by the power consumption of applications. Except the application's characteristics, such as UI and special functionality that improve user experience, end users can also take power consumption into consideration.

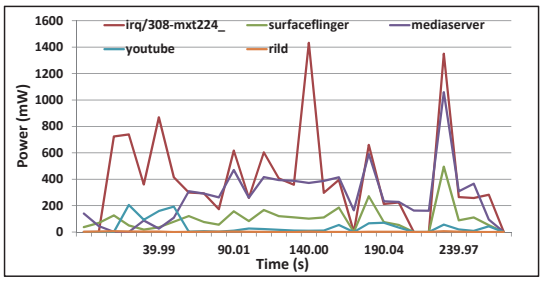
Take browser as an example, assume users want to install *Opera* on their device. They can simply send the type of device and application name *Opera* to the Bugu server. Table 4.4 lists several applications' data stored in the server, and the browser part will be returned to the users.

Browser	Application Power	Game	Application Power	Health	Application Power	Reading	Application Power
Opera	123.42	NinJump	141.73	Instant Heart Rate	65.96	Kindle	86.34
Dolphin	162.15	Temple Run	142.75	Lose Tt	83.55	Daily Bible	131.23
Firefox	304.63	Cut the Rope	149.12	Cardiograph	92.26	Audible	158.95

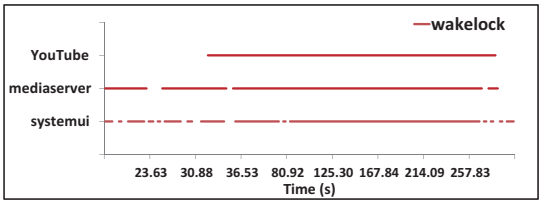
Table 4.4: The comparison of applications power consumptions. (The power unit is mW.)



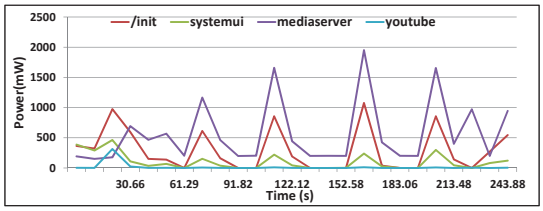
(a) Event information of YouTube running on Nexus S.



(b) The power variation information on Nexus S.



(c) Event information of YouTube running on Motorola Xoom.



(d) The power variation information on Motorola Xoom.

Figure 4.3: The comparison of YouTube event and power information.

Noted that the power data is calculated under the general usage situation. In this experiment, we chosen six popular websites including *cnn, espn, amazon*, opened them one by one and each time scrolled down to see all the information. We can see that *Opera* consumes less power than *Firefox*, which makes it more competitive. To figure out the behind reasons, we analyzed the event information and raw power log data of *Opera* and *Firefox*, the results show that their CPU power has big difference. *Firefox* may do more processing and calculation to improve user experience, further analysis about their power behavior can be found in Section 4.4.3.

Application Developer Scenario On one hand, the Bugu server provides related applications' power data for application developers to compare. On the other hand, developers can

get event information from the Bugu client, which gives the optimization direction from power consumption aspect. In this section, we use video application as an example to show how Bugu works. In our experiment, the new application developed is *YouTube*.

To include the influence that application may bring to the system, the Bugu server not only provides each applications' information, but also gives other four most power consumption processes of each application and compares the union of them. Thus, there are six processes compared in Figure 4.4. The data is collected from Nexus S, and we can see that system processes which support our applications consume much more power than the application itself. For instances, *systemui* is responsible for drawing the user interface, *mediaserver* provides sound and other support for media. From the perspective of the target application, *YouTube* is in a good situation, its power is lower than others. Figure 4.3 presents the event information and power variation of *YouTube* on both Nexus S and Xoom. These information helps developers deeply understand the power issue. For event information, the x-axis is time; the y-axis is the processes that generate these events. The recorded events includes wakelock, sensor, screen, etc. We only found wakelock information in this scenario, the *mediaserver* process appeared in both devices, and $rild$, $surfaceflinger$ occupy wakelock on Nexus S while *systemui* on Xoom side. *YouTube* also occupies the wakelock for a long time as showed in Figure 4.3a and 4.3c, developers can analyze their code to improve the wakelock utilization, for instance, release wakelock in app pause state. Figure 4.3band 4.3d demonstrate power variation of processes which occupy the event or has high power consumption, *YouTube* consumes high power when we start the application, while *mediaserver* periodically reached the high point. After analyzed the resource usage information in the beginning of *YouTube*, we found it transmitted network packets and dealt with user inputs (e.g. touch, click). No other abnormal data detected. Another reason for such high power consumption is the preparation system did for starting new activity. So if the developers want to optimize *YouTube*, they should focus more on handling user inputs efficiently and balancing data downloaded.

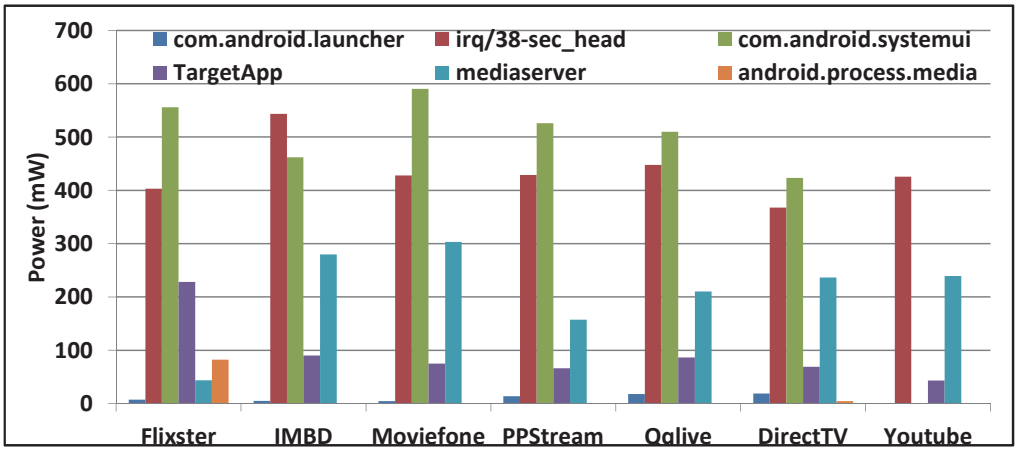


Figure 4.4: The power comparison of seven video applications.

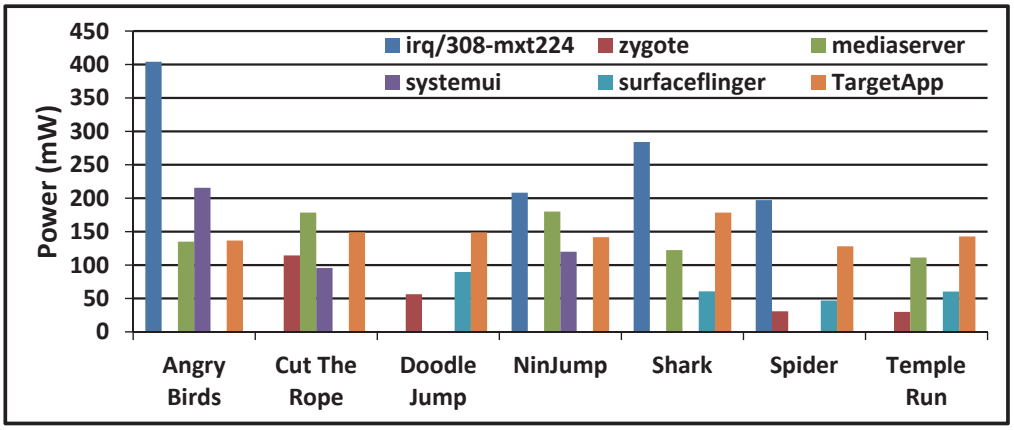
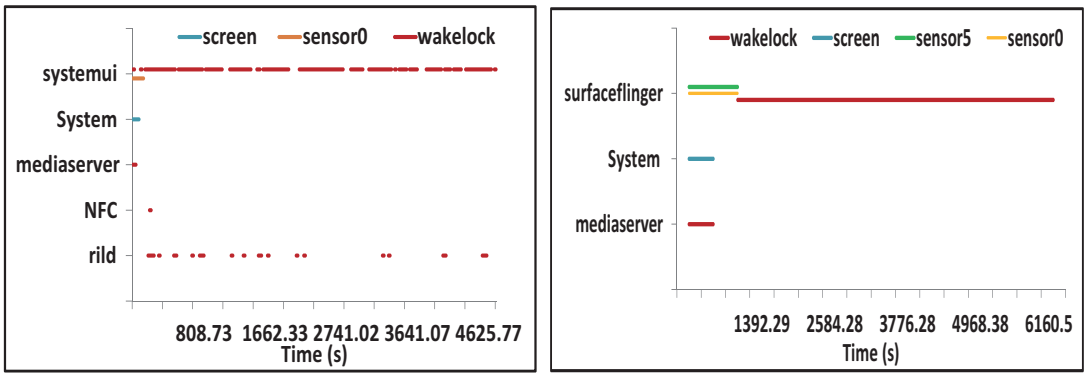


Figure 4.5: The power comparison of seven games.

System Developer Scenario

System developers care more about the whole system power consumption, not a particular application or hardware component. Bugu provides power information of all processes running in the system, which exactly helps them to know the whole picture. From previous experiments, we observed that system processes consume much more power than target application itself. To show it is a common issue, we did another experiment on game applications. We evaluated 7 popular games: Angry Birds, Cut The Rope, NinJump and so forth. Figure 4.5 demonstrates the power consumption of each game and several corresponding system processes on Nexus S. We can see that system processes, such as *irq/308-mxt224*, *mediaserver* and *zygote*, consumed much power, they were not negligible comparing with our target applications.

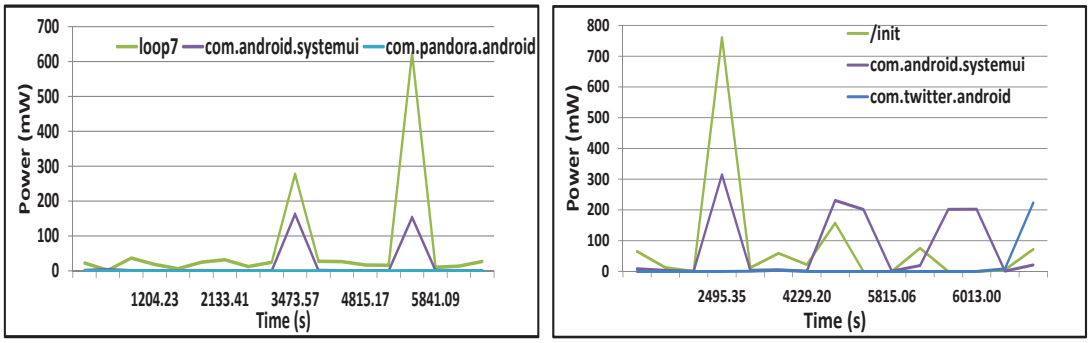


(a) The event information on Nexus S.

(b) The event information on Xoom.

Figure 4.6: The comparison of devices event information under "sleep" mode with no application running.

Except active applications, background applications are also a main concern for system designers. We did several experiments to show how applications and the system behave in sleep mode. In the experiments, we first tested the situation that only system processes exist and no applications opened. From Figure 4.6 we notice that *surfaceflinger* occupied the wakelock almost all the time on Xoom, while *systemui* and *rild* dominated on the Nexus S. *surfaceflinger* and *systemui* work on the user interface drawing and rendering part, *rild* is responsible for the phone service. These processes acquired and released wakelock continuously, which make the processor can hardly get the chance to work in C states. Comparing with Nexus S, Xoom can last much more longer after one fully charge. To present the real case when users use these devices, We did the experiments with applications running in the background. The most common situation it represents is when users go to sleep, their mobile devices are in sleep state without exiting all opened applications. Figure4.7 shows the power variation of top three power consuming processes of Nexus S and Xoom under "sleep" mode with unclosed applications. Before we put the devices into "sleep" mode, we opened *facebook*, *twitter*, *youtube*, *angrybird* and *pandora*, and played with each of them for a few minutes. From the figure we know that most applications' power are low, while system processes still consume a lot. Hence, system developers should focus more on optimizing these background processes and services.



(a) The power variation on Nexus S. (b) The power variation on Xoom.

Figure 4.7: The comparison of devices power information under "sleep" mode with background applications.

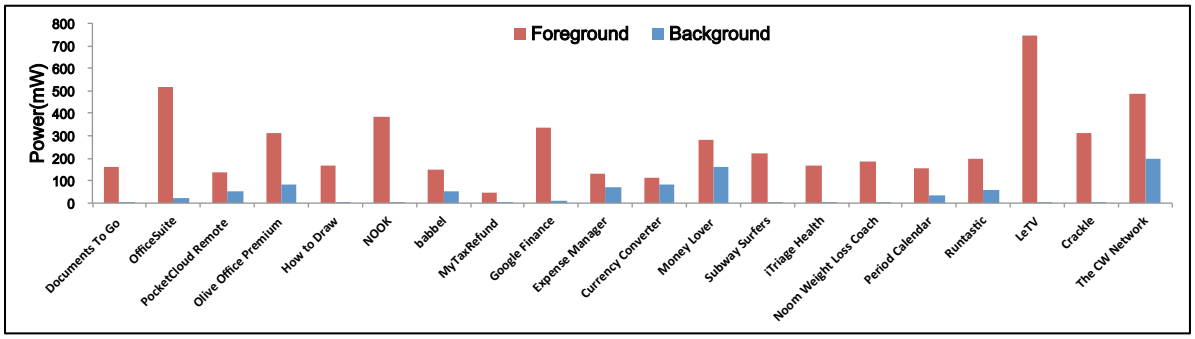


Figure 4.8: The comparison of applications background and foreground power consumption.

4.4.3 Applications Power Information Analysis

In this section, we analyze mobile applications' power data in both foreground and background situation. For each case, we describe applications' total power and distribution among main hardware components.

Apps Run in Foreground: We first introduce foreground scenario. In the experiments of 100 applications, their power ranges from 20 mW to over 700 mW. 10% of them consumes less than 50 mW, 50% is less than 200 mW. The average power is 227 mW, and 20% is greater than 335 mW in our dataset. It is reasonable that the power varies so much. Pdf reader will run longer than Angry Bird with the same battery capacity. Intuitively, the power consumption of applications in the same category should be in the same level. To further prove the statement, aside from video and game applications presented above, we also took the Education, Health

and Fitness applications' data into consideration. In these four categories, Education apps consume less power than the other three categories; only the power of NYTimes greater than 200 mW. Most of Media and Health apps power are within 300 mW and 200 mW respectively. In a specific category, the applications' power also varies. The power difference between Temple Run 2 and Speed Skater is as high as 300 mW. For applications produced by the same company, the difference is smaller as "Talking" series (Talking Angela 320 mW, Talking Ben 350 mW, Talking Tom Cat 410 mW) suggest.

To figure out where the power goes, we analyzed the detailed power information logged by the Bugu client which contains main hardware components' power dissipation for each application. We summarize two metrics which are important factors to reflect component power information: $Numberof Appearance$, it is defined as number of applications use the component over total number of applications, and $PowerRatio$, which equals the percentage of the component consumed power over total application power. According to our experiment results, CPU is used in all applications and its average $PowerRatio$ is the highest in the components we considered. This means most of the time the CPU dominates the applications' power. 13% and 20% applications use GPS and Audio respectively, and they contribute nearly 20% power to the applications. Although there are only 14% applications in our dataset play with Video module, the average $PowerRatio$ is as high as 61%. After focusing on high video power ration application, we found that video power is much higher than CPU power except two applications that their video power and CPU power are almost equal. Hence, when applications play video, its main power dissipation very likely transfers from CPU to Video.

Apps Run in Background: For background applications, we classify them as two categories: idle background and active background. The former represents the applications that will stop working and enter suspend status when in background. Active background means the applications that still have activities even in background, such as download applications and music applications.

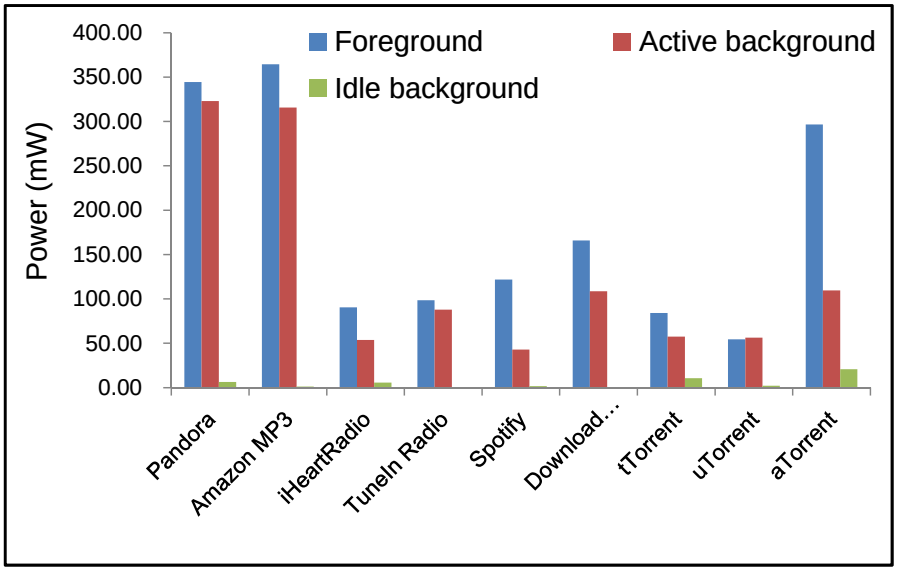


Figure 4.9: The comparison of applications power consumption in foreground, active background and idle background.

Idle Background: The idle background situation is common for most applications, especially for media apps and games. Figure 4.8 demonstrates background power and foreground power of 21 applications in our dataset. 60% of the applications, their background power is less than 50 mW, and two of them are over 100 mW. The background power varies from 1.5 mW to 190 mW. The applications are listed by their category. Similar with the foreground case, the background power of applications which are in the same category also varies; Office Suite background power is 20 mW while Olive Office Premium reaches 80 mW. For applications with high ratio of background power to foreground power, like Expense Manager, we found their power is dominated by CPU power consumption.

When applications go to idle background state, users move their focus to the new foreground application. Except maintaining the status in case they will run again in a short time, they should occupy resources as less as possible. Hence, an energy efficient application should reduce their background power consumption and maintain the ratio of background power to foreground power in a relatively small range.

Active Background: Some applications still active and function normally when they are

in background. For example, we open Pandora to listen to music and at the same time we check emails or read news in foreground. In that situation, we claim that Pandora is in *ActiveBackground* state. For this kind of applications, they complete most of their work in active background situation.

In the experiments, we choose five popular applications from Music and Audio category: Pandora, iHeart Radio, Amazon MP3, TuneIn Radio and Spotify, and four download applications: Download Manager, tTorrent Lite, uTorrent and aTorrent. Amazon MP3 randomly played local songs and radio applications played several stations, four download applications downloaded a 325 M video file. Figure 4.9 describes their power dissipation in foreground, active background and idle background situations. When the applications enter active background situation, their power dissipation is less than foreground case and most of them only decrease a little. The power consumption of uTorrent in the two cases are almost the same, Pandora's power reduce 20 mW which is about 5% of total power. For idle background situation, six of their power are less than 5 mW, aTorrent's idle background power is also less than 6% of total foreground case power. Spotify and aTorrent decrease around half of the power when enter active background case, the user experience of the two applications did not change, there was no visible delay to play music and download the video. The possible explanation may relate to other functionality suspended in background.

Real Apps Case Studies

We analyzed several applications' power information, which includes: *Pandora*, *iHeartRadio*, *Facebook*, *Firefox*, by tracking resource usage information through Bugu. With the consideration of system processes and the comparison of similar applications' data, we revealed some underlying reasons of high power situations. In the experiments, we installed target applications in a clean OS and the logged power is the whole system power. Since we only run the target application in foreground and suspend all background processes (e.g. Google Services), any system power variation is mainly caused by the target application.

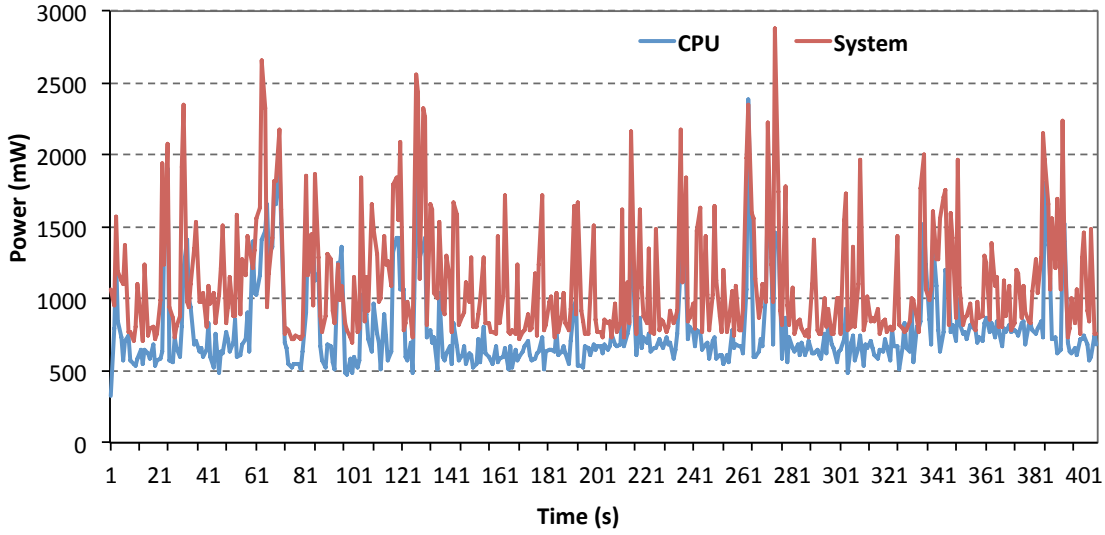


Figure 4.10: The system and CPU power information of Pandora.

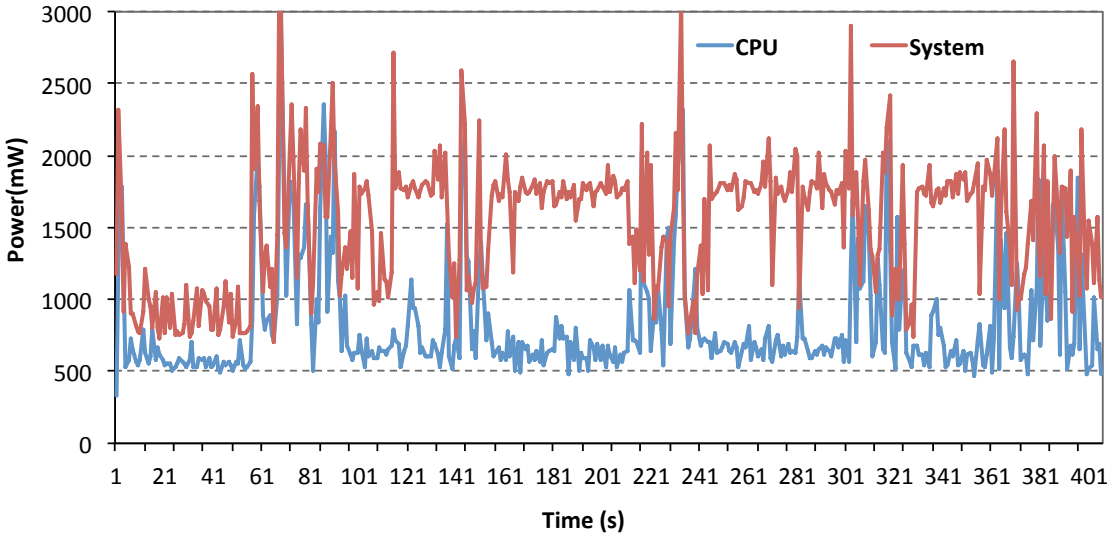


Figure 4.11: The system and CPU power information of iHeartRadio.

In the Music and Audio category, the most popular applications are *Pandora* and *iHeartRadio*. Figure 4.10 and 4.11 demonstrate their CPU and system power variation when listen to music. By comparing the two applications' system power, we noticed that the power of *iHeartRadio* was higher than *Pandora*'s when playing music. The CPU power in the two applications was reasonable as the high power situations were caused by handling user input, such as changing channel/song, and network transmission. To find the root cause that lead to high

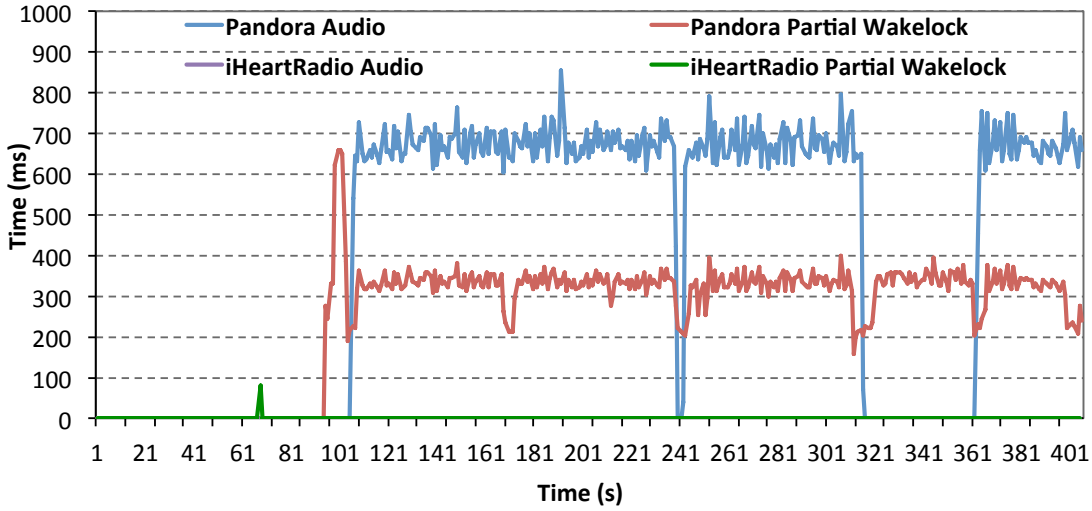


Figure 4.12: The information of wakelock and audio time for Pandora and iHeartRadio.

system power of *iHeartRadio*, we first detailed compared the resource usage information which includes audio, video, wakelock, CPU time and network packets of the two applications. Figure 4.12 shows their audio and wakelock time, the audio time of *iHeartRadio* was all 0. Hence, when we calculate the application power, the result of *iHeartRadio* was less than *Pandora*'s (showed in Section 4.4.3) as the audio power and wakelock power of *iHeartRadio* were almost 0. However, it did not illustrate the high system power of *iHeartRadio*. Next, we analyzed the information of all processes running in the system. Aside from Bugu, system (uid:1000) and the target application, the active process was mediaserver (uid:1013) in both cases. Because the audio time is logged in *MediaPlayer.java* in Android OS, we think *iHeartRadio* did not use build-in player program to communicate with mediaserver, which causes high system power consumption when playing music. The resource usage information of *Douban Artists* further proved the statement since the trend of audio time and system power were similar with *Pandora*'s. The wakelock data of *iHeartRadio* and *Douban Artists* was almost the same, and the high wakelock usage in *Pandora* was mainly the result of frequent advertising.

Social network application becomes the main platform that keeps people in touch in today's society. In our experiments, we checked the latest news of friends and posted the status with and without photo. The power variation of *Facebook* and resource usage information are

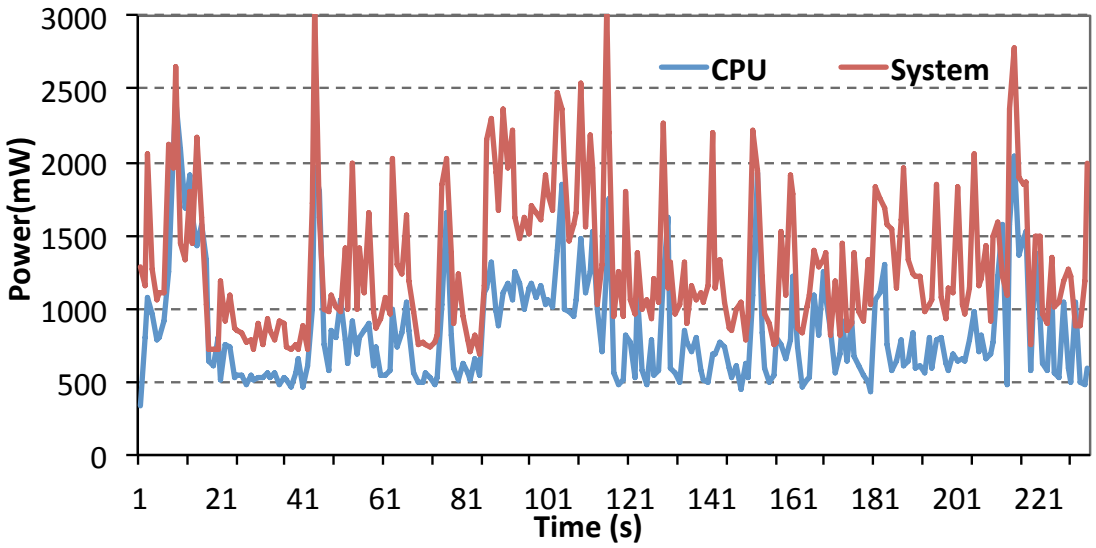


Figure 4.13: The system and CPU power variation of Facebook.

demonstrated in Figure 4.13 and Figure 4.14 respectively. At the starting of the application and dealing with the user inputs, CPU power dominated the whole system power. There is a high power period from 81s to 115s, and it is caused by taking a photo as showed in Figure 4.14 (*Facebook Full Wakelock* and *Facebook Accelerometer* overlapped). When we prepared to post status, the location process became active for several seconds and it used *Particial Wakelock* and GPS. Users may share their location in the posted status. As the result, the corresponding system power was increased a little bit. The same situation can be also found in *Twitter*, the location process appeared and the system power increased. When we posted a status with photo, *Twitter* delegated the job to Android default application *Gallery* while *Facebook* handled by itself. On the aspect of the system power, the two approaches are similar although the wakelock and accelerometer were used by different processes.

For browser applications, we analyzed *Firefox* and compared its data with *Opera*'s as the high power consumpsion of *Firefox* demonstrated in the previous section. We opened several popular webpages. On the perspective of system processes, the situations of the two applications were similar. The particial wakelock's time of mediaserver and location process occasionally increased, they were not actually in active state. The wakelock time of *Google Search*

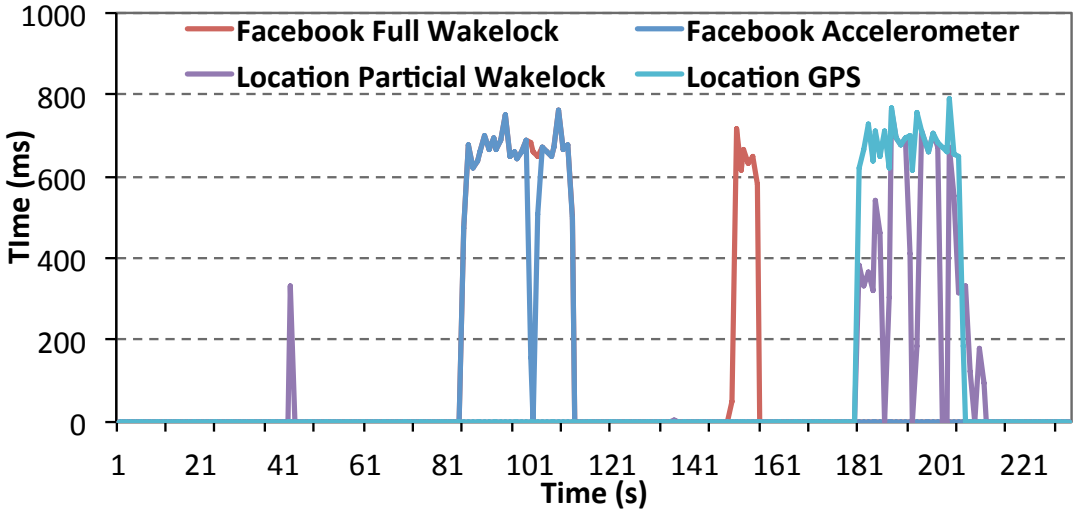


Figure 4.14: The part of the system resource usage information when playing with Facebook.

Box also increased, which was more frequent than in other applications, such as *YouTube* and *TempleRun*. For application itself, CPU power dominated the whole power. Aside from user inputs, network activity also causes high CPU usage. Figure 4.15 illustrates the system power and packets information when *Firefox* was in foreground. The peak points of high packets transmission correspond to high power consumption. There are a lot of times that packets were over 10000, while the situation happened much less in *Opera*'s case. Hence, we think it is the main reason for high power consumption of *Firefox*. For download applications, high packets transmission may help save energy since the system can go to sleep state after the job is done. However, it is not hold for frequent user interactive applications as the interval time between two tasks (user inputs) is not always longer enough for system to switch to the sleep state.

4.4.4 Bugu Accuracy

There is no ground truth for application level power consumption. Hence, to analyze the accuracy of Bugu, we focused on the whole system power. Figure 4.16 demonstrates the measured power and estimated power for several popular applications. We used a BK Precision programmable power supply [54] to power up the smartphone, which provides a constant voltage of $3.8V$ and records current data. We calculated the system power, which is listed as measured power, based on the current information. The estimated power is calculated and

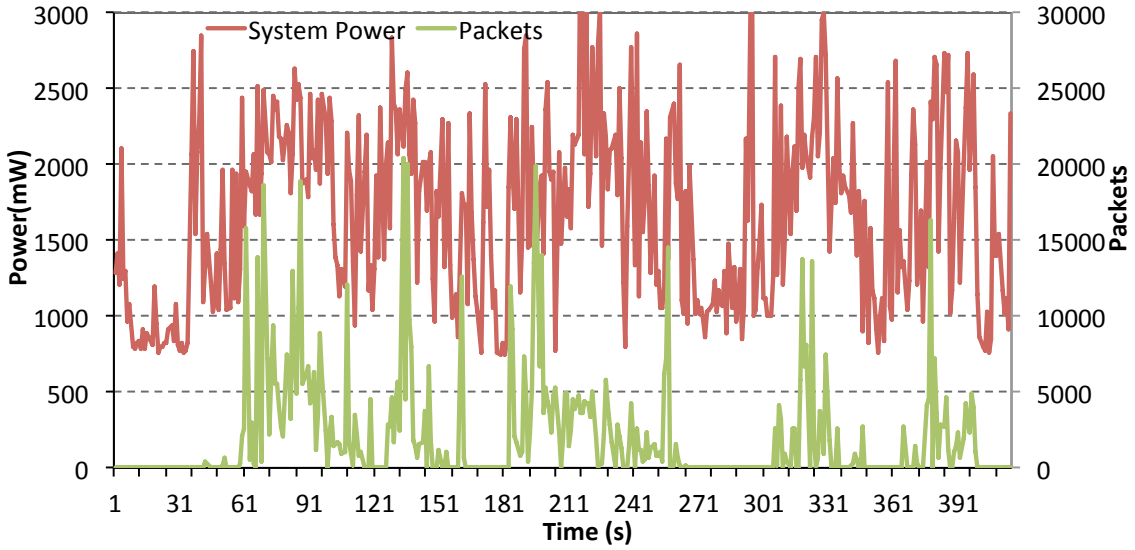


Figure 4.15: The system power and packets information of Firefox browser.

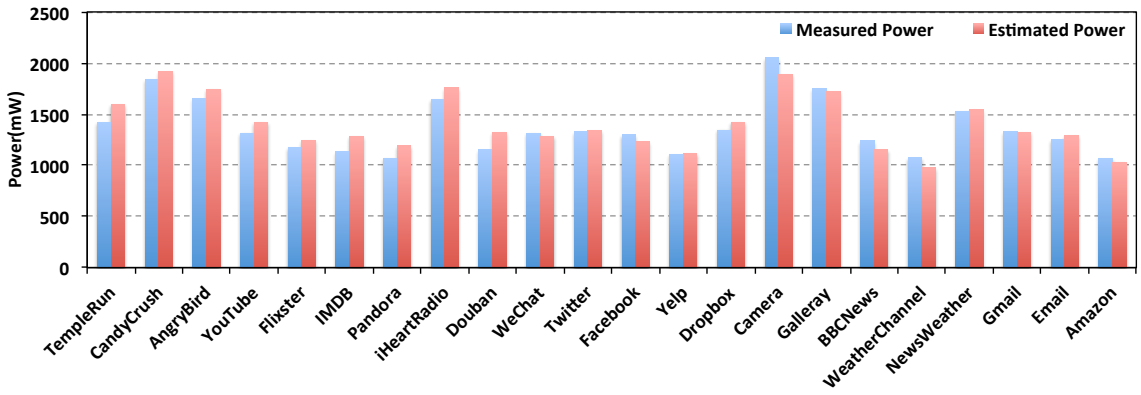


Figure 4.16: The comparison of measured power and estimated power for popular applications. logged by Bugu. For game, music and video applications, the estimated power is greater than hardware measured power; for social and utility applications, the most results from Bugu is equal or less than the measured power. The average error rate of Bugu for total system power is 5%.

4.4.5 Bugu Overhead

The overhead of Bugu is mainly caused by the power profiler and event monitor. The data processing is done when the user wants to read an experiment record. When we did experiments described above, we also recorded the power consumption of Bugu. The power

consumption of Bugu is around 5mw to 10mw, which accounts for 2.52% of the foreground application power consumption on average. Moreover, we compared the system power with and without Bugu. The power results were calculated by attaching the power meter to the battery. For the situation that no active foreground application exists, Bugu causes 200mW extra system power. Because Bugu samples resource usage information once per second, it stops the CPU and system to stay in a low power state and lead to such amount of system power overhead. In real measuring cases, there is always a "target" application running, the average extra power Bugu generated on the system level is around 100mW. Compare with 1000mW to 1500mW whole system power, the overhead is acceptable.

4.5 Implications

4.5.1 Radio Service

Our experiments show that *rild*, which is the daemon of Android radio service, generates a lot of wakelocks even when the device is not active. Even though the power consumption of this service is not high, it keeps the processor active and consumes a large amount of energy. These wakelocks are generated during processing unsolicited commands, such as network status change, SMS notify, USSD (Unstructured Supplementary Service Data) notify and signal strength or time changed. Among these unsolicited commands, some of them, such as SMS notify, are important to users. However, we do noticed that a large amount of unsolicited commands, such as signal strength change, received are not highly required. To design an energy-efficient Android radio layer interface, we should reevaluate the structure of unsolicited command processing part by filtering part of the commands. In this way, we can increase the chance of making the device work in the "real sleep mode". Another approach is putting the long lasting service to a low power coprocessor, so that the coprocessor can handle part of the data processing without waking up whole system.

4.5.2 Hardware Interrupts

In addition, we observed that several processes, such as *irq/308-mx224_* and *irq/38-sec_head* consumes a large amount of energy when we ran several applications. *irq/308-mxt224_* is the threaded interrupt handler for the touchscreen controller. Different with traditional cell phones and normal computer systems, current mobile devices have much more sensors to supply various functionality to users. These sensors generate a large amount of hardware interrupts and consume a large amount of energy. Aside from *rild*, the sensor related processing can also be delegated to the low power coprocessor. We argue that we should revisit the design of interrupt handling part for current mobile operating systems since the design of hardware platform is totally different now.

4.5.3 Energy-efficient Applications

As our experiments in Section 4.4.3 show, applications' power consumption varies a lot. In low battery status, users can stop some unnecessary applications to save energy for phone service. Besides, the power consumption of applications in the same category can be very different even though the functionality of them are the same. That means, it is possible to develop energy efficient applications without influencing user experience. In detail, there are two directions that we can look into. Aside from the main hardware components, other parts such as DSP, sensors are also needed to be used in an energy-efficient way. For example, video applications may use video module more than CPU since video module power consumption dominates the whole application's power. Another direction is to improve applications' background situation. From the experiment results, some applications in the background are not really suspended. Considering the user behavior that they usually put applications in background rather than kill them, these applications may generate big influence on the system energy. Hence, when the developers implement applications, they should reevaluate the background case and decrease the power consumption as much as possible.

4.5.4 System Power Management Design

One important goal that we design operating system is to protect the hardware from misuse by applications. However, some power management APIs, like wakelock, are not used efficiently and they can cause big energy issue. We think the APIs for application design should be reevaluated from energy saving angle.

The energy consumption of screen, processor, radio and wifi accounts for about 95% of the whole system energy consumption. Among these devices, it is hard to decrease the power of screen, which accounts for about 50%, through many kinds of system level energy-efficient strategies. In addition, the space to decrease the power of radio and wifi is low if the users need to use them. Even if we could filter some of the unsolicited commands, we cannot make radio work in low power mode. Thus, it is nearly inevitable to design an energy-efficient strategy that can drop the energy consumption of the system significantly. So, we claim that there is no chance to solve the power problem for mobile devices with a single energy-efficient strategy. The mobile operating system needs a group of energy-efficient design strategies to work together to accomplish this goal.

4.6 Summary

In this chapter, we built a power profiler Bugu and analyzed almost 100 mobile applications power behavior using it. Bugu is composed of a server side which provides power information of different applications, and a client side that analyzes power and event information for specific applications. We implemented Bugu on Android platform and evaluated its accuracy (95%) and overhead. We showed the case studies of finding the root causes of large power consumption, for example, the overuse of wakelock. The analysis of applications power information is useful for many energy/power related researches on mobile devices, and the implications derived from the observations point out several potential optimization directions.

In the next chapter, we focus on studying the potential energy benefit from modern heterogeneous platforms which are the trend of future mobile devices. Specifically, we will compare

heterogeneous platform and homogeneous platform and figure out how to take advantage of heterogeneity.

CHAPTER 5 HETEROGENEOUS PLATFORM ENERGY EFFICIENCY ANALYSIS

Heterogeneous multi-core platforms, e.g., ARM's big.LITTLE, are a promising trend to improve the performance and energy efficiency of future mobile systems. However, the immediate benefits and the challenges to take advantage of the heterogeneity are still not clear. In this chapter, we present our experiences about the energy efficiency of the two big.LITTLE heterogeneous platforms: ODROID XU+E and ODROID XU3. We quantified compared them with homogeneous platforms through multiple benchmarks. We analyzed the scheduling impact on the energy consumption of the heterogeneous platforms as well as the migration cost. Based on the results, several insights related to hardware, application and system design are derived.

The remainder of the chapter is organized as follows: Section 5.1 presents the motivation and introduction. We illustrate the two heterogeneous platforms and our experiment setup in Section 5.2. The detailed case studies are demonstrated in Section 5.3, which compared the heterogeneous and homogeneous platforms and analyzed the impact of scheduling and migration cost. Following that, we discuss a list of insights in Section 5.4.

5.1 Introduction

As the result of the dark silicon issue and increasing demand of specialized components, heterogeneity becomes more and more important and leads the trend of future devices' development, especially for mobile platforms. Heterogeneity is a general concept that may refer to CPU/GPU computing architecture, mixed types of accelerators and so on. The advantage of heterogeneous platforms is that they can improve energy efficiency while maintaining performance[34, 110]. To evaluate their benefits, previous work usually leverages DVFS to simulate different types of CPUs [111]. With the emerging of the ARM big.LITTLE processor and Samsung Exynos 5 Octa system-on-chip (SoC) [112], we have the opportunity to explore and exploit real heterogeneous hardware platforms.

In this chapter, we undertake the following questions:(1) *Compared with homogeneous*

Table 5.1: The specifications of the two platforms.

	ODROID XU+E (XUE)	**ODROID XU3 (XU3)**
Generation	The first generation	The second generation
SoC	Samsung Exynos 5410	Samsung Exynos 5422
big CPU	quad-core Cortex-A15 CPU (800 MHz to 1600 MHz)	quad-core Cortex-A15 CPU (1200 MHz to 2000 MHz)
LITTLE CPU	quad-core Cortex-A7 CPU (500 MHz to 1200 MHz)	quad-core Cortex-A7 CPU (1000 MHz to 1500 MHz)
L1 Cache	32 kB/ 32 kB	32 kB/ 32 kB
L2 Cache	2 MB on big CPU, 512 KB on LITTLE CPU	2 MB on big CPU , 512 KB on LITTLE CPU
Memory	2 GByte	2 GByte
Key Features	CPU hotplug, Cluster switching	CPU hotplug, Heterogeneous multi-processing (HMP)

platforms, how much energy can be saved in heterogeneous platforms? We want to know the capability of heterogeneous platforms. (2) *what is the impact of scheduling from energy aspect?* As the different types of processor exist, the scheduling algorithm takes the responsibility of choosing proper processor for different workloads. Both of the benefits can be gained from the correct scheduling and penalty of the improper scheduling are important. And finally (3) *what is the migration overhead on performance and energy?* Applications have different phrases (e.g. loading content, waiting user input, etc.) and scheduler needs dynamically migrate workload to proper cores in runtime. In this situation, migration overhead is one of the key factors to decide when and which core to migrate. All of the three questions directly influence the benefits we can get from heterogeneous platforms.

To answer these questions, we picked the two ARM big.LITTLE platforms, ODROID XU+E (XUE) and ODROID XU3 (XU3) [113], as experimental platforms to study. Both XUE and XU3 have heterogeneous cores but provide different control strategies. XUE offers cluster switching and XU3 supports heterogeneous multi-processing [112]. The system and component level power information are collected to analyze their energy behavior under various cases. By analyzing the results, we can further understand the characteristics of the big.LITTLE platforms and wisely use them.

5.2 Experiment Setup

To practically investigate heterogeneous platforms, we did experiments on two generations of ARM big.LITTLE platforms produced by Hardkernel [113]. Their specifications are presented in Table 6.1. ARM's big.LITTLE processor is a single-ISA heterogeneous multi-core processor which contains two types of cores: Cortex-A15 and Cortex-A7. Data is shared between the clusters via the CCI-400 cache coherent interconnect to achieve seamless migration. The first generation, denoted as XUE, provides only cluster switching mode, that is, either the big cluster or the LITTLE cluster is active. The cluster switching is achieved by modifying the CPU frequency. On the contrary, XU3, the second generation, supports heterogeneous multi-processing (HMP) which is a core level migration and all the eight cores can be active at the same time. We can dynamically set the CPU affinity using *taskset* command to migrate thread between the different cores. For homogeneous platforms, we disable the cluster migration on XUE and control the platform runs on big (LITTLE) cores only to simulate A15 (A7) platform. Both of XUE and XU3 feature four separated current sensors to measure the power consumption of big core cluster, LITTLE core cluster, GPU and memory in realtime. To get the system power data, we use a BK Precision programmable power supply [54] to power up the platforms, which provides a constant voltage of 5 V and a maximum current of 5 A. It logs the current at a sample frequency of 4 Hz.

5.3 Case Studies

In this section, we investigated the two big.LITTLE platforms by comparing their power and energy consumption with homogeneous platforms' results under various applications. The experimental benchmarks include mobile applications and high-performance synthetic benchmarks. Moreover, we analyzed the scheduling impact and migration overhead of heterogeneous platforms to study how to get the benefits of heterogeneity.

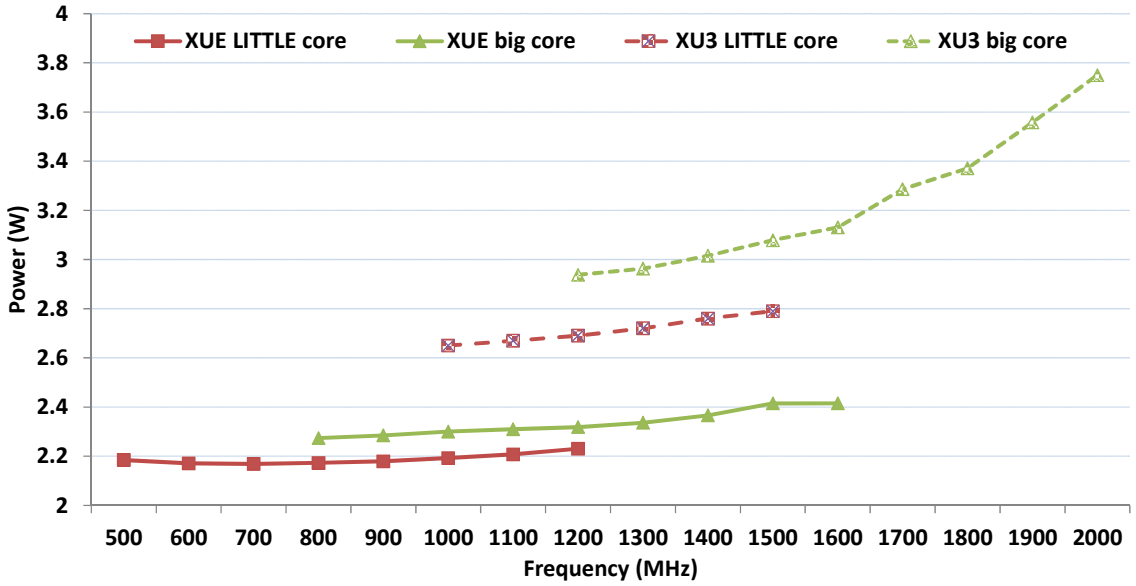


Figure 5.1: The active idle power of the two platforms under each frequency.

5.3.1 Active Idle Power

The short battery life of mobile devices is one of the main motivations that lead to the development of big.LITTLE architecture which promise to deliver peak-performance capacity at significantly lower average power. The energy is saved by running on LITTLE (energy efficient) cores when the system is in idle state. Most time mobile devices, like smartphones, are in sleep state, so we first investigate the active idle power difference between homogeneous and heterogeneous platforms to see how much we can save. Here the active idle represents the situation that system is awake but no application is running.

Figure 5.1 presents the active idle power of the big and LITTLE cores at each frequency in the Android OS. The power refers to the whole device's power, not the core level power. The LITTLE core power of XU3 is measured when all the big cores are disabled. At the same frequency, the big core active idle power is always greater than LITTLE core power, and the difference is around 0.1 W for XUE and 0.25 W for XU3. The range of LITTLE cores' power is very small, less than 0.2 W. While the big cores are sensitive to frequency change, so that it can provide corresponding high performance. Assume a smartphone's idle time is 8h, the battery voltage is 4 V. Compared with *A15 Only* homogeneous platforms, the heterogeneous platform

with LITTLE core can save as much as 200 mAh. The system power difference of XUE and XU3 mainly caused by other components on the boards, not the result of the heterogeneous cores, GPU or memory which we can measure power directly.

One of the well-known power saving approaches in multi-core systems is core offlining [81]. We evaluated the influence of disabling CPU cores on the two platforms. Due to the page limit, we illustrated XU3's results here and the behavior of XUE is similar. *The offlining did not work well within CPU clusters.* The power was always 3.18 W when we disabled 1 to 3 big cores. Then, the power decreased sharply from 3.18 W to 2.68 W since all the big cores were disabled and the package was idle. For LITTLE core cluster, the system power decreased slightly (from 2.68 W to 2.64 W) when the number of active cores decreased.

5.3.2 Benchmarks

As we presented previously, heterogeneous platforms improve system energy efficiency in the active idle case. Next we will investigate their performance and energy information under various workloads.

We first analyze mobile applications since heterogeneous platforms already appear in the mobile market. Six mobile applications/benchmarks are chosen: *YouTube* and *Castle Master* represent the two popular application categories: Video Player and Game. *RAR* is a compression program and *PhotoShop (PS)* is a photo editing tool. They are "heavy" workloads that appear on mobile devices. The last two applications are *BBench* [114] and *GFXBench*, they are browser and GPU benchmarks used to stress the system and are not commonly used in daily life.

Figure 5.2 shows their component level energy consumption. "Other" refers to the power used by components rather than CPU, GPU and memory. Its value is total system power minus the measured five parts (A7, A15, memory, GPU, active idle). The heterogeneous platform data were collected from XUE, because XU3 has higher CPU frequency which may interfere comparison results. To keep the figure readable, we proportionally adjusted the time within

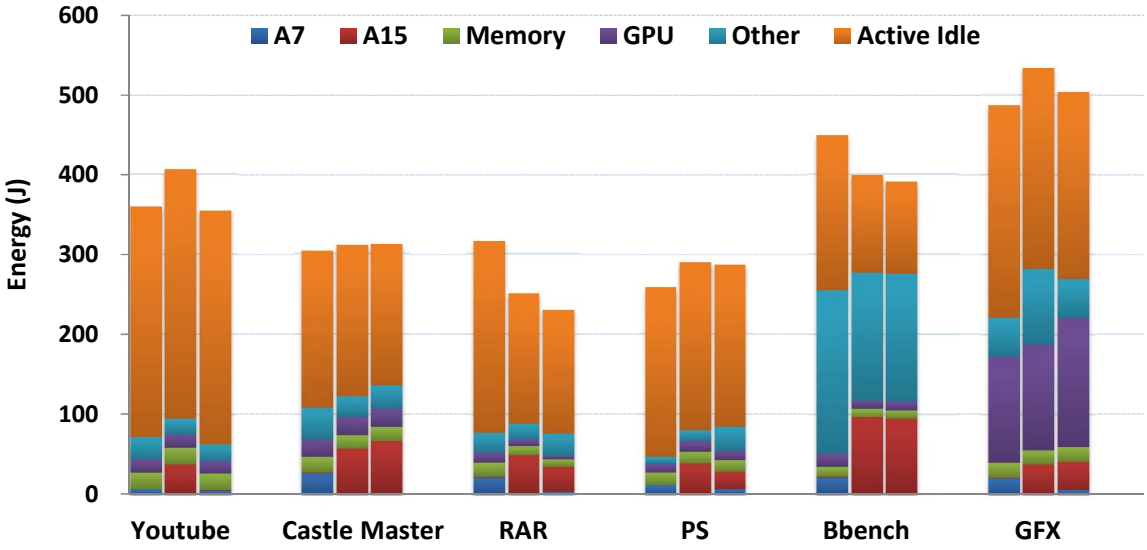


Figure 5.2: The component level energy information of mobile applications on A7 Only, A15 Only and XUE platforms (left to right).

each application. From the viewpoint of energy, *A7 Only* platform consumes the least energy in most cases. However, with the consideration of user experience, which is an important factor for mobile devices, not all the energy optimal configurations are suitable. There is an obvious delay when *PhotoShop* and *Castle Master* run on A7 cores. Except *YouTube*, XUE leveraged A15 cores in all the applications. The A7 energy in XUE is not obvious due to its low power and less active time, but compare the total system energy of XUE and *A15 Only* platform, the contribution of A7 is identifiable. From the results, *we argue that performance sensitive applications merely contribute to the energy efficiency improving due to performance constrains, while light workloads benefit most on heterogeneous platform.* For applications that stress a specific component, like *GFX* which GPU power consumptions are similar in the three platforms, the heterogeneous CPU cores are not very helpful. However, as most of mobile workloads are periodical [110], there is still considerable energy saving potential on heterogeneous platforms.

Nowadays, multi-core architecture is widely-used in computer systems and it improves parallel applications' performance a lot. Both XUE and XU3 contain quad-core CPU, so we focus on the parallel benchmarks in the following examples, such as sysbench and NAS Parallel

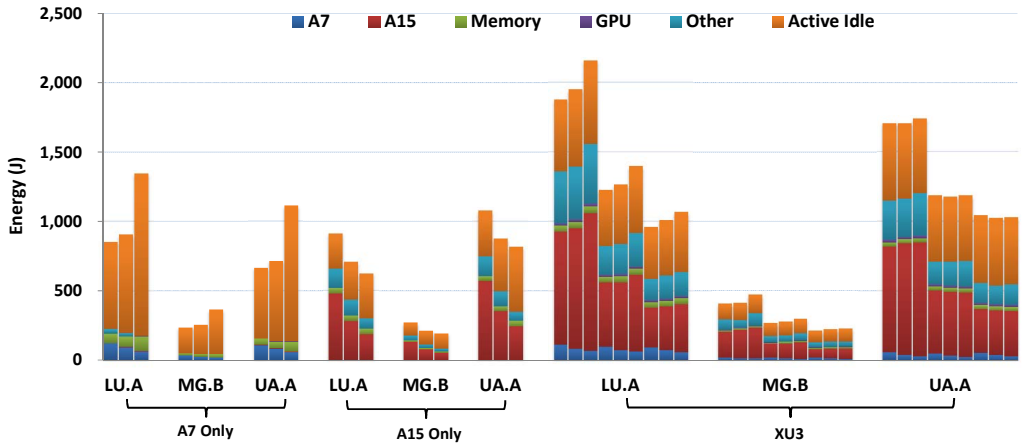


Figure 5.3: The component level energy information of NPB benchmarks on the platforms. The frequencies (left to right) are 1200, 1000, 500 MHz on A7 Only platform and 1600, 1200, 800 MHz on A15 Only platform. The frequencies (left to right) for big core and LITTLE core on XU3 are 2000&1400, 2000&1200, 2000&1000, 1600&1400, 1600&1200, 1600&1000, 1200&1400, 1200&1200, 1200&1000 MHz.

Benchmarks (NPB) [115], to evaluate their performance and energy behaviors.

As we mentioned above, the XUE platform allows four cores (big or LITTLE cores cluster) to be active, while the XU3 platform supports all eight cores working concurrently. Hence, we tested the energy consumption of the sysbench CPU benchmark which calculates the prime numbers that are smaller than 10000 on the two platforms with 1 to 8 threads. The core frequency is fixed (big:1200 MHz and LITTLE:1000 MHz) to eliminate the potential interference from DVFS. From the perspective of the total system energy consumption, XUE consumed less energy than XU3 in 1 to 4 threads cases. As the number of threads increased, the XU3 platform became more energy efficient because it can leverage the four more LITTLE cores. The execution time on the two platforms is almost the same and the average difference is 0.6 s when we ran 1 to 4 threads. After that, XU3's time continually decreased until reaching eight threads. Although the active idle power of XU3 is greater than XUE's, its system energy becomes smaller as the result of the decreased time. *Hence, similar with homogeneous multi-core systems, the number of active threads should match the number of cores to take advantage of multi-core architecture.*

Parallel benchmarks usually run on highest frequency to get better performance, but the

high frequency usually is not the energy optimal configuration. To analyze the parallel benchmarks' energy behavior on heterogeneous cores, we measured the energy consumption of the NPB benchmarks under several frequency settings. Figure 5.3 lists three representative benchmarks' results. During the benchmarks' running time, there is no migration exist in XUE, so we leverage XUE to get homogeneous platforms' results and the compared heterogeneous platform in the examples is XU3. We used 4 threads to run on XUE and 8 threads to run on XU3 so that we can leverage all the available cores. On the homogeneous platforms, the energy optimal setting is 800 MHz in *A15 Only* platform and 1200 MHz in *A7 Only* platform. For the heterogeneous platform, the optimal configuration is always the smallest frequency on the big core, while the optimal LITTLE core frequency depends on the programs (LU.A prefers high frequency, UA.A has no obvious preference). *Since the active idle power is varied a lot based on the big core frequency, which can not be compensated by the speedup, the smallest frequency on the big core is always the optimal energy choice in both platforms.* For component level energy consumption, the "Other" part increased with the increase of frequency, especially on big cores. Besides, we can see that there is a trade off between CPU energy and memory energy, the memory energy consumption increases with the decreasing of CPU frequency.

From the perspective of performance, all benchmarks' execution time decreased with the frequency increase on the homogeneous platforms, while the results varies on XU3. For LU.A and MG.B cases, their running time mainly depended on the LITTLE core frequency. The time of UA.A was decided by both big and LITTLE cores. Hence, there is an obvious energy difference caused by the LITTLE core when the big core frequency is fixed for LU.A and MG.B, while the same situation was not found in UA.A. *Compared with the homogeneous platforms, the performance on heterogeneous platform is not directly proportional to the frequency since it has two types of cores working at the same time which makes the synchronization issue becomes more significant.*

5.3.3 Impact of Scheduling

With the more components we can control, the potential to achieve better energy efficiency becomes higher, while the complexity to find the optimal configurations also increases [116]. Multiple applications can run concurrently, the scheduling becomes the crucial part that directly influences the performance and energy efficiency. Assuming that there are two processes running in the system, is it better to put them on the same core or different cores?

We took LU.A and UA.A as examples. The benchmarks were compiled as one thread program, and each time we run two instances (e.g. LU.A.1 and LU.A.2) under different core configurations. The results are shown in Figure 5.4, the most energy efficient choice is putting the two processes on two big cores and the second optimal option is using one big core. It is reasonable as the big core consumes less energy than the LITTLE core when there is only one process. The energy consumption of leveraging both big and LITTLE core is similar with running on two LITTLE cores. The energy difference between the two settings and the optimal configuration is as high as 30% of the *1b+1L* case. From application's viewpoint, the LU.A benchmark is more sensitive to the LITTLE core. Compared with UA.A, the energy consumption of LU.A is very different under cases with and without LITTLE core. One of the reasons is that LU.A uses the CPU more intensively than UA.A.

From previous synthetic benchmark examples, we can see that the key factors of scheduling are the number of threads, the number of cores as well as workload preferred core characteristics. Next, we analyze the scheduling impacts of application benchmarks with different phrases.

The scenario included *BBench* which simulates user web browser behavior and music player that run on the background. There was a two-second pause after each page loaded to mimic reading behavior. Hence, two phrases exist in the example: the loading phrase which requires big core and the pausing phrase which prefers run on LITTLE core. The music player should always run on LITTLE core based on its requirement. The default scheduling method

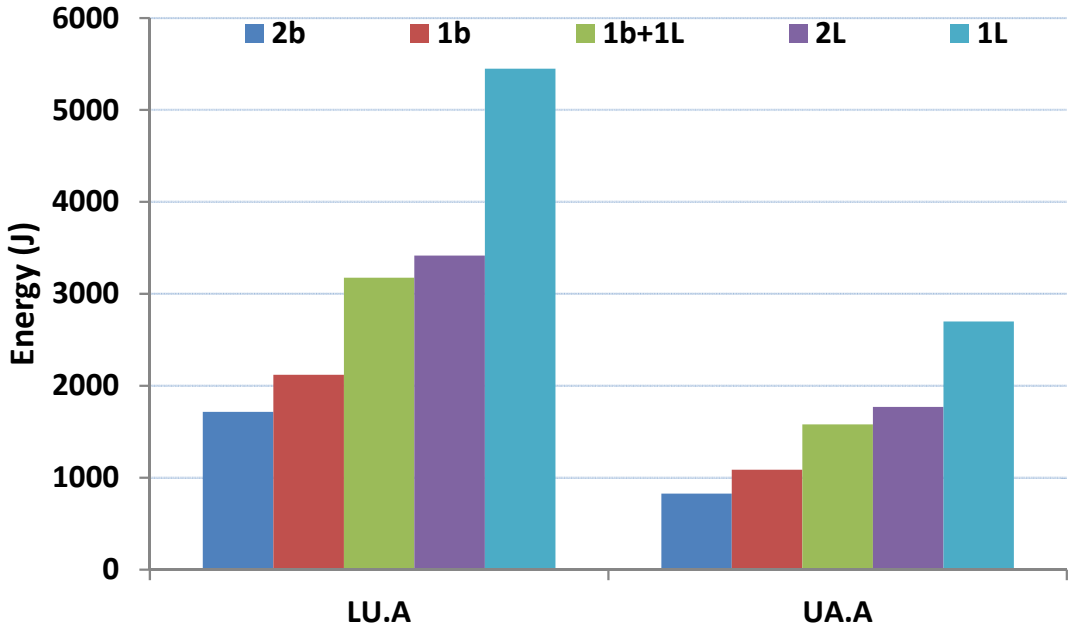


Figure 5.4: The energy consumption of LU.A and UA.A under different configurations, big core frequency is 1600 MHz and LITTLE core frequency is 1200 MHz. *b* and *L* represent big core and LITTLE core respectively.

was indirectly controlled by offline CPU cores, so the workload is forced to run on the available cores. Figure 5.5 presents the energy consumption under different configurations. The homogeneous *A7 Only* case consumes the least energy since its power consumption was small and there was no much speedup in other cases. In the heterogeneous cases, the energy difference of *XU3 Default* and *XUE Default* are mainly caused by the difference in their idle power. The 5% difference of *XU3 Default* and *XU3 2b+2L* is the results of the workload's demands on big cores. We omited the *XU3 1b+1L* case in the figure, because its time was twice of others which leads to the highest energy consumption and no comparability.

The principle of scheduling is providing enough resources to workloads. The consequence of improper scheduling depends on workload's characteristics. Compared with synthetic benchmarks, the phrase-based applications are more tolerant.

5.3.4 Migration Cost

The advantage of the ARM big.LITTLE architecture is that we can choose the proper type of cores to use in the runtime to save energy. In this section, we investigated the migration cost

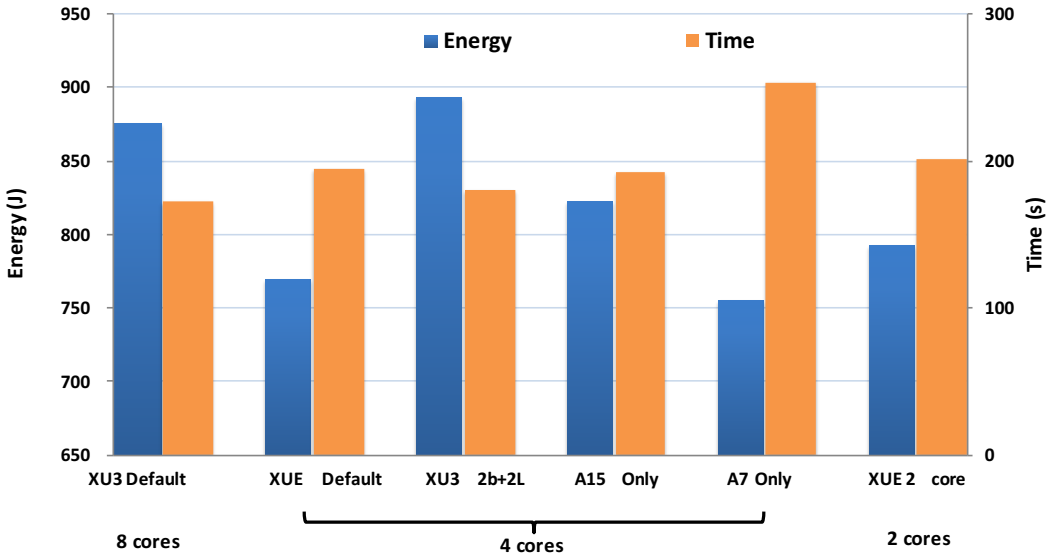


Figure 5.5: The energy consumption of BBench under different configurations with default scheduling. b and L represent big core and LITTLE core respectively.

from the aspects of performance and energy.

To evaluate the migration cost, the basic approach is to compare the results with the ground truth which does not include migration overhead. However, the ground truth is almost impossible to measure on real devices. Hence, we take the theoretical value as the baseline, which is calculated as follows: Assume migration overhead is zero, the whole workload W running only on big (LITTLE) core requires T_{bW} (T_{LW}) time and E_{bW} (E_{LW}) energy. Then the workload completed in the unit time is: $w_b = \frac{W}{T_{bW}}$ for big core, and $w_L = \frac{W}{T_{LW}}$ for LITTLE core. Suppose in the migration enable case, the workload is migrated after running on big core for t_b seconds or running on LITTLE core for t_L seconds. So the time ratio of running on big core and LITTLE core is $t_b : t_L$. Assume the total big core running time is $t_b * k, k \in \mathbb{R}$, then

$$W = w_b * t_b * k + w_L * t_L * k = \frac{W}{T_{bW}} * t_b * k + \frac{W}{T_{LW}} * t_L * k$$

So $k = \frac{T_{bW} * T_{LW}}{t_b * T_{LW} + t_L * T_{bW}}$, and the theoretical running time is:

$$t_{Th} = t_b * k + t_L * k = \frac{(t_b + t_L) * T_{bW} * T_{LW}}{t_b * T_{LW} + t_L * T_{bW}}$$

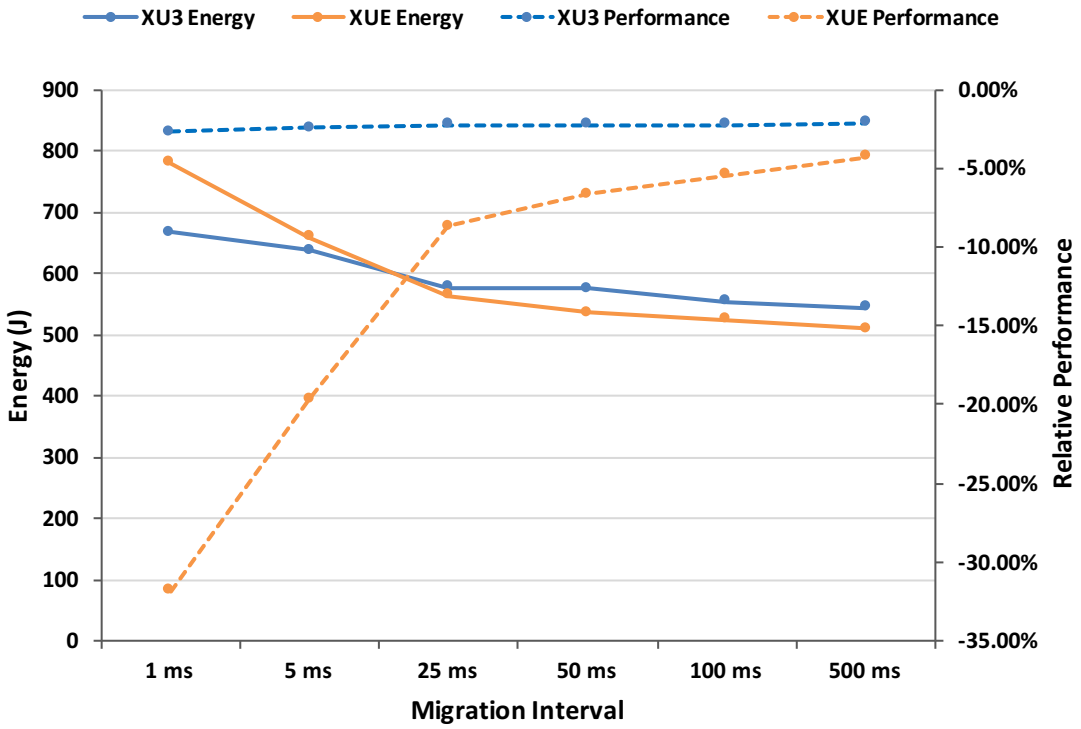


Figure 5.6: The energy and performance information of sysbench CPU benchmark in different migration interval cases.

Similarly, the theoretical energy is:

$$E_{Th} = \frac{E_{bW}}{T_{bW}} * t_b * k + \frac{E_{LW}}{T_{LW}} * t_L * k$$
$$= \frac{E_{bW} * t_b * T_{LW} + E_{LW} * t_L * T_{bW}}{t_b * T_{LW} + t_L * T_{bW}}$$

Figure 5.6 illustrates the migration impact on energy consumption and performance for sysbench CPU benchmark. The workload is migrated every x milliseconds, with x varying from 1 ms to 500 ms. It belongs to the $t_b : t_L = 1 : 1$ case. We can see that there is obvious energy and performance cost for XUE and energy cost for XU3 with the migration interval decreases. The potential reason for the different performance trend of XUE and XU3 is the cache coherence issue on XUE which leads to cache miss after cluster migration. Hence, we make the big cores offline/online after each migration on XU3 to redo the experiments. This time the relative performance is similar with the XUE's results. With the migration interval becomes longer, the energy and performance of the two platforms are more and more stable.

Table 5.2: Migration cost for CPU benchmark.

Type	t_b **:** t_L	**XUE T (s)**	**XUE E (J)**	**XU3 T (s)**	**XU3 E (J)**
Typ_M	1:0	94.89	379.56	94.31	386.67
Typ_M	0:1	219.41	548.52	219.16	727.61
Typ_{Th}	1:1	132.48	430.57	131.87	489.24
Typ_M	2:2	132.67	443.12	132.11	507.3
Typ_M	6:6	133	434.91	132.25	493.29
Typ_{Th}	1:2	152.64	457.92	152.06	544.37
Typ_M	2:4	153.52	463.63	152.54	562.87
Typ_{Th}	2:1	117.03	409.6	116.42	447.04
Typ_M	4:2	117.56	420.86	116.91	460.62

Table 5.2 presents the results in different $t_b : t_L$ cases with larger migration intervals. Typ_M stands for measured value and Typ_{Th} represents theoretical value. The first two rows that represents the situations that we only run the benchmark on big core (first row) and LITTLE core (second row). As Table 5.2 reports, the time delay is negligible since the theoretical time and measured time are very close, while the energy is different to some extent. Compare the two platforms, the energy consumptions of XU3 are greater than XUE's as the result of high active idle power. The same situation on the energy overhead aspect, take the 4:2 case as an example, the cost for one migration is 0.56 J in XUE and 0.68 J in XU3. In the two cases that the time ratio is 1:1, the results indicate the 6:6 case consumed less energy than the 2:2 case since the migration times is less in the 6:6 case. In a word, there is an energy cost to migrate between big and LITTLE cores. It is not always good to choose the optimal setting since the migration may not worth the price if the workload changes frequently.

5.4 Insights

Based on the results from case studies, we derived a list of implications and grouped them into the following three categories:

Hardware Design: *Provide fine-granularity power control to further decrease idle power.* Based on the comparison of homogeneous and heterogeneous platforms, low idle power is the key factor that improves the energy efficiency on heterogeneous platforms. To save more energy, devices should provide fine-granularity power control and decrease components' power

coupling so that each component can stay in low power mode freely. For example, CPU offlining in the platforms does not impact the system power very much unless all the cores in the package are idle. We can provide independent power supply to each core to effectively reduce CPU power.

Extend heterogeneity to multiple components. ARM big.L-ITTLE provides heterogeneity in CPU level, while the power dissipation of CPU only occupies part of the mobile system's power. The heterogeneity can be applied to other components to better serve users' requirements for different applications and save the energy at the same time.

Application Design: *Increase the usage of thread level parallelism while pay attention to synchronization.* Similar with homogeneous multi-core platforms, to benefit from the increasing number of cores, applications should improve their thread level parallelism. Most of the popular mobile applications only use one or two cores and there is little chance that they can leverage four cores. Comparing the XU3 platform with XUE, there is no obvious performance improvement in CPU intensive phrase, so as energy since the speedup cannot compensate the increased power. Compared with homogeneous platforms, the synchronization issue in multi-thread applications are more significant since the heterogeneous cores have different capabilities and the performance may easily be influenced by the workload runs on the LITTLE core. Take LU.A as an example, the execution time of a four-thread case on the *A15 Only* platform with big cores at 1200 MHz is smaller than a eight-thread case that runs on XU3 with big cores at 1200 MHz and LITTLE cores at 1000 MHz.

Operating Systems Design: *Schedule tasks to the right core at the right time.* The scheduling algorithm directly affects the energy consumption and performance of systems. On the application level, we found that different applications have different optimal configurations, for example, LU.A prefers big core at 800 MHz while UA.A works better on LITTLE core at 1200 MHz. If the workload is scheduled to a wrong type of core, the energy difference can be as high as near 30%. On the system level, there are usually multiple applications running concur-

rently. The system needs to detect non-CPU intensive phrases and schedules them to LITTLE cores with the consideration of migration cost, so that the total system energy is saved. Moreover, the energy consumption is not the only aspect that we care about during scheduling, user experience should also be considered.

5.5 Summary

Heterogeneous platforms lead the trend of future devices, especially in the mobile market. We found that heterogeneous platforms indeed have great potential for energy saving which mostly comes from idle and low workload situations, however, there are several steps that should be taken seriously by the community, including hardware vendors, application developers, and operating systems designers, to maximize the potential of heterogeneous platforms.

With the mobile devices become more and more powerful, the power density increased sharply, which makes the thermal control becomes one of the design bottlenecks. In the next chapter, we will discuss the energy-based thermal control policy Falcon. It decreases cooling energy within the same thermal constrain and adapts to various ambient temperature.

CHAPTER 6 FALCON: TEMPERATURE AWARE THERMAL CONTROL POLICY

With the development of the devices, thermal control becomes the issue that limits the hardware design since the reliability of electronic hardware components is closely related to their operation temperature. Besides, the blooming of the Internet of Things (IoT) requires devices work properly in different environment, which makes the thermal control even harder. System designers normally rely on fans to cool the system because it is cheap and easy to implement. Hence, in this chapter, we take IoT smart gateway with fan as our example to discuss the thermal control policy in modern devices. The problem of fan is that the over cooling will waste energy and cause noise. So, we propose Falcon, an ambient temperature aware thermal control policy for smart gateways. Falcon leverages a runtime thermal prediction model and takes the ambient temperature into the consideration to tune the fan speed proactively. In addition, the two environment adaptive mechanisms applied in Falcon help us achieve the same thermal control ability in different ambient temperature environment, especially in high ambient temperature cases.

The outline of this chapter is as follows: Section 6.1 gives a brief introduction. Then, we describe the thermal prediction model and Falcon in Section 6.2. Following that, Section 6.3 presents the model validation and the evaluation of Falcon. Finally, Section 6.4 summarizes the chapter.

6.1 Introduction

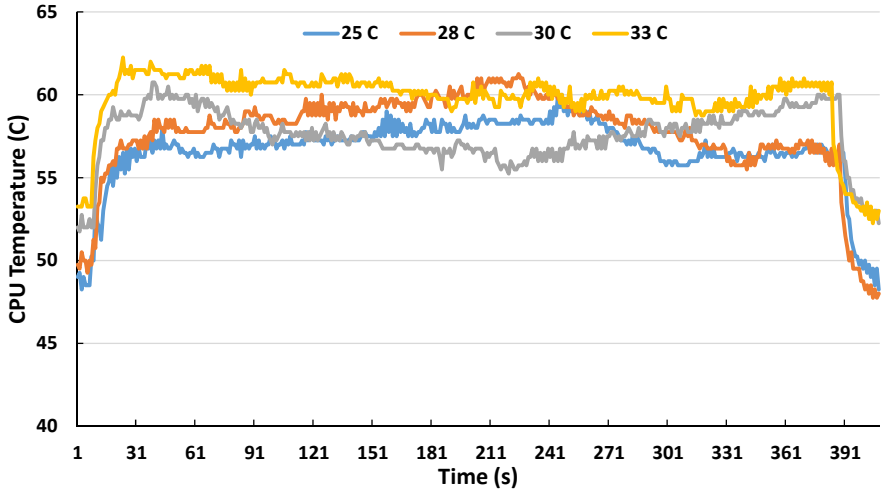
We are entering the Internet of Things (IoT) era. According to the prediction of the International Data Corporation [117], the global IoT market will grow from $655.8 billion in 2014 to $1.7 trillion in 2020 with an annual growth rate of 16.9%. Connectivity is the key in IoT, which makes gateways become important components. Intel has announced its IoT Gateway platform which provides seamlessly data transmission [118]. With more and more devices connected, we need intelligent gateways that can offload the computation and communication tasks of IoT equipments. We envision that smart gateways will be the trend in the near future.

Gateways can run their own operating system with suitable computation capability. Hence, in this dissertation, we take Odroid-XU+E [46], ARM big.LITTLE platform, as the reference smart gateway and evaluate our ideas on it.

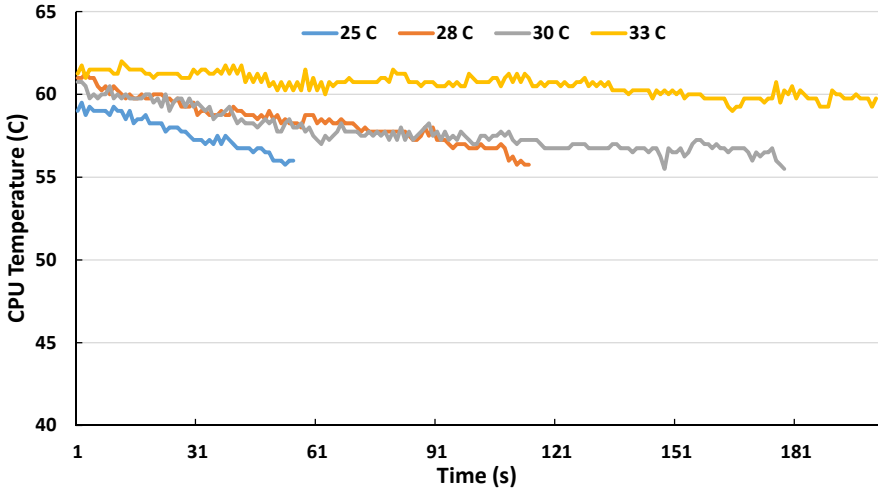
The reliability of gateways is very important. The devices connected to it will be "missing" from the Internet if the gateway is down. However, the processor failure rate *exponentially* depends on the operation temperature [41]. When the temperature reaches the trip point, the system performance will decrease by thermal throttling or the device directly shut down. To avoid hurting the performance, we think a CPU fan will be applied on gateways as the cooling method.

Gateways can be deployed indoor for smart home or outdoor for collecting environmental sensor data. For indoor gateways, fan noise is an inevitable issue, which we have not paid enough attention. Since it is proportional to the fifth power of the fan speed, a lower speed will be very helpful. Besides, lower speed also saves a lot of power as fan power is a cubic function of fan speed [119, 89]. For example, the total system active idle power of the reference gateway is 2.83 W when the fan runs at 50% speed, while the fan power occupies 16.5%.

Compared to servers which are located in air conditioning rooms, the ambient temperature of gateways varies a lot. A good cooling approach should be feasible in most situations. However, the current default fan control policy does not work well in high ambient temperature cases. Figure 6.1a demonstrates the CPU temperature variation of MistBench (See Section 6.3.1) under 25°C, 28°C, 30°C and 33°C cases. The fan was set to full speed when the CPU temperature reaches 63°C and idle in other cases. There is an obvious temperature increasing in the 25°C and 28°C cases, while in 30°C and 33°C cases, the temperature reached the maximum value at the beginning. The ambient temperature greatly influences the device's thermal behavior. To clearly illustrate the fan cooling efficiency in each experimental case, we compare the CPU temperature decreasing phrases in Figure 6.1b. We can see that to reach the low temperature 56°C, the fan working time was very different (from 61 s to 181 s) and the



(a) The comparison of CPU temperature variation.



(b) The comparison of fan cooling effects.

Figure 6.1: The CPU temperature of MistBench in different ambient temperature cases. 33°C case did not make it. For high ambient temperature, the fan cooling efficiency decreased a lot. The default fan control policy which set the same threshold without considering ambient temperature can not effectively cool the device.

To reduce the noise and apply a lower fan speed under the same thermal constraints for indoor gateways, we first built a thermal prediction model based on the previous CPU temperature, the ambient temperature and the power dissipation of the four main components (big cores, small cores, memory and GPU) of the reference gateway. We proposed Falcon, a thermal control policy, according to the model and compared its performance with the default fan con-

figuration. Our evaluation result shows that Falcon can save 4.85% of the total system power and reduce fan speed in 34% of the time on average. Moreover, to address the high temperature environment thermal control issue, we first experimentally illustrated the impact of the different ambient temperature from the temperature and power dissipation aspects. The active idle power increased 6% in the 40°C case compared with the 10°C case and the CPU peak temperature is proportional to the ambient temperature. Then we proposed a refined ambient temperature aware thermal control policy which improves fan cooling effect by modifying the temperature threshold and restrics generated heat through decreasing CPU frequency. The thermal control ability is evaluated. Compared to the default fan configuration, Falcon can reduce average temperature in 2.7-10 °C in high temperature environment.

6.2 Ambient Temperature Aware Thermal Control

In this section, we illustrate the thermal prediction model that estimates the CPU temperature. Based on the model, the ambient temperature aware thermal control policy (Falcon) is proposed to reduce the power dissipation and noise under the same thermal constraints. In addition, comparing with default fan configuration, Falcon provides better cooling effects in high ambient temperature situations. Note that we use degrees Celsius (°C) when discuss temperature in this section.

6.2.1 Thermal Prediction Model

Fan cooling is one of the forced convection approaches to transfer heat. In the heat transfer theory [120], thermal resistance measures the ability to transfer the heat, thermal capacitance refers the heat an object can store. Based on the energy balance [41], the thermal model is described as:

$$C\frac{dT}{dt} + \frac{T - T_{amb}}{R} = Q$$

where C is the thermal capacitance (J/°C), R is the system thermal resistance ($°C/W$), T is the CPU temperature (°C), T_{amb} is the ambient temperature and Q is the total generated heat (W). In this dissertation, we assume forced convection (fan) is the main heat transfer method

and ignore other approaches such as natural convection and heat conduction. Heat transfer by convection is expressed by Newtons law of cooling:

$$Q_{conv} = hA(T_s - T_\infty) \tag{6.1}$$

where h is the convective heat transfer coefficient ($W/(m^2 \cdot^\circ C)$), A is the surface area (m^2). Fan's thermal resistance $R_{fan} = \frac{1}{hA}$. h measures how effectively a fluid transfers heat and it is determined by many factors include fluid density, viscosity, velocity and so on. Theoretically, h can be estimated from Nusselt number (Nu), Reynolds number(Re) and Prandtl number (Pr) [120].

$$Nu = \frac{hL}{k}, \qquad Re = \frac{\rho VL}{\mu}, \qquad Pr = \frac{\mu C_p}{k}$$

where parameters are: characteristic length L (m); fluid thermal conductivity k (W/m·°C); mass density ρ (kg/m^3); velocity V (m/s); viscosity μ (N·s/m^2); and specific heat capacity C_p (J/kg·$^\circ C$). In the fan convection case, $Nu = \alpha Re^{1/2} Pr^{1/3}$, α is a constant. Since the ambient temperature influences the air density and our goal is to choose a proper fan speed, we keep these two variables and others can be treated as constants. So,

$$h = f(\rho, V) = \beta(\rho V)^{1/2}$$

where β is a constant. Hence, we can choose the proper fan speed based on the required heat transfer coefficient value.

By leveraging the approximation that $\frac{dT}{dt} \approx \frac{T(k+1)-T(k)}{\Delta t}$, where $T(k)$ stands for the temperature of the kth unit time, the discrete-time thermal model can be expressed as following:

$$T(k+1) = (1 - \frac{\Delta t}{C} hA)(T(k) - T_{amb}) + \frac{\Delta t}{C} \alpha' P(k) + T_{amb} \tag{6.2}$$

where Δt is the sampling interval. Heat is generated by the power, so we use $\alpha' P$ to represent the Q, where α' is constant. Under the same sampling rate, $\frac{\Delta t}{C}\alpha'$ is constant, $1 - \frac{\Delta t}{C} hA$ changes with h. In our reference platform Odroid-XU+E, there are thermal sensors to log big cores temperature and four power sensors for little core cluster, big core cluster, memory and

GPU. Hence, we use matrix $P = [P_{little}, P_{big}, P_{mem}, P_{GPU}]^T$ in the thermal model. The simplified version is:

$$T(k+1) = A'(T(k) - T_{amb}) + B'P(k) + T_{amb}$$

where A' is a function of fan speed and air density (as $A' = 1 - \frac{\Delta t}{C}hA$) and B' is a matrix that contains constant parameters.

6.2.2 Thermal Control Policy Falcon

Generally, assume the thermal constraint is referred as T_{ref} , our goal is to select a proper fan speed under the situation that $T(k+1) \leq T_{ref}, \forall k \in \mathbb{N}$. According to the thermal prediction model, we can estimate future temperature. One of the important parameters is T_{amb}. Since gateways may be deployed in different environments, we cannot assume T_{amb} is constant. Fortunately, the ambient temperature value can be calculated from build-in CPU thermal sensor. In the idle steady state, $\frac{dT}{dt} = 0$, so $T_{amb} = T - RQ$. The heat generated in the system idle state are the same since the idle power does not change much. Besides, the system thermal resistance is also fixed without enabling the fan. Hence, the value of RQ is irrelevant with environmental temperature. The system just needs input ambient temperature once, then it calculates and stores the RQ value which can be used to estimate T_{amb} next idle time.

The current default fan configuration is monitoring thermal sensor data and operating at specific speed (20%, 50% or 100% of full speed) if the temperature is over the pre-defined thresholds. However, it does not work very well for IoT gateways. For indoor gateways, noise is a key factor that needs to be considered. With our dynamic fan speed control policy Falcon, the thermal prediction helps reduce fan overuse cases. In addition, the default fan configuration is not suitable in high ambient temperature environment as presented in Section 6.3. Falcon addresses the issue through two approaches. First, we decrease the T_{ref} to improve fan cooling effect. Falcon adopts ${T_{ref}}'$ as

$${T_{ref}}' = T_{ref} - f(T_{amb})$$

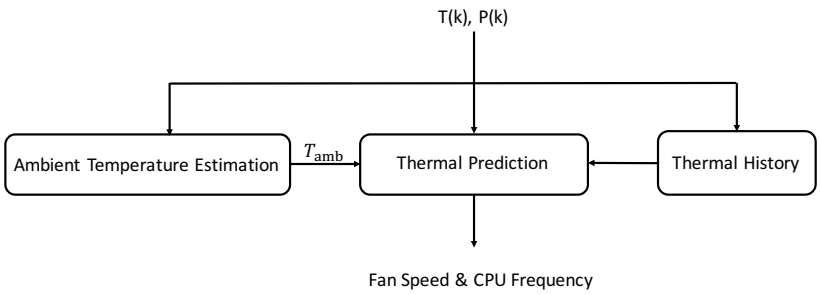


Figure 6.2: The Falcon overview.

From equation 6.1 we see that if we use the same temperature threshold, the transferred heat decreases in high ambient temperature cases since the ΔT is smaller. Hence, to maintain the temperature difference, the threshold temperatures should be decreased accordingly. Another approach to deal with the high ambient temperature environment is controlling generated heat. Fan cooling effect is bound by the heat it can transfer in full speed case. So if the fan reaches its limitation, we should consider reducing generated heat to maintain the thermal constraint. Specifically, we adjust CPU frequency to reduce power, thereby controlling generated heat.

Figure 6.2 illustrates the basic flow of Falcon. The Ambient Temperature Estimation module calculates current ambient temperature every time when the system is in idle state. Thermal History is used to help decide which cooling approach will be used in high ambient temperature environment. If the CPU temperature quickly reaches the threshold T_{ref}, we should adjust frequency to control generated heat. On the contrary, if there is an obvious temperature increasing process, we can reduce the temperature threshold to enable fan early so that the previous accumulated heat can be transferred before it contributes to increasing CPU temperature. In that way, the system performance will not be influenced too much. Based on the thermal and power data, the proper fan speed and CPU frequency (if possible) are estimated. To simplify the calculation and decrease the overhead, we prefer to identify the parameters A' and B'. Besides, a table of discrete fan speed v_i and its corresponding value of $A_i{}'$ is kept in the system as a reference. In Falcon, the $T(k+1)$ is calculated from the lowest fan speed to the highest fan speed, the one that satisfy $T(k+1) \leq T_{ref}$ is chosen.

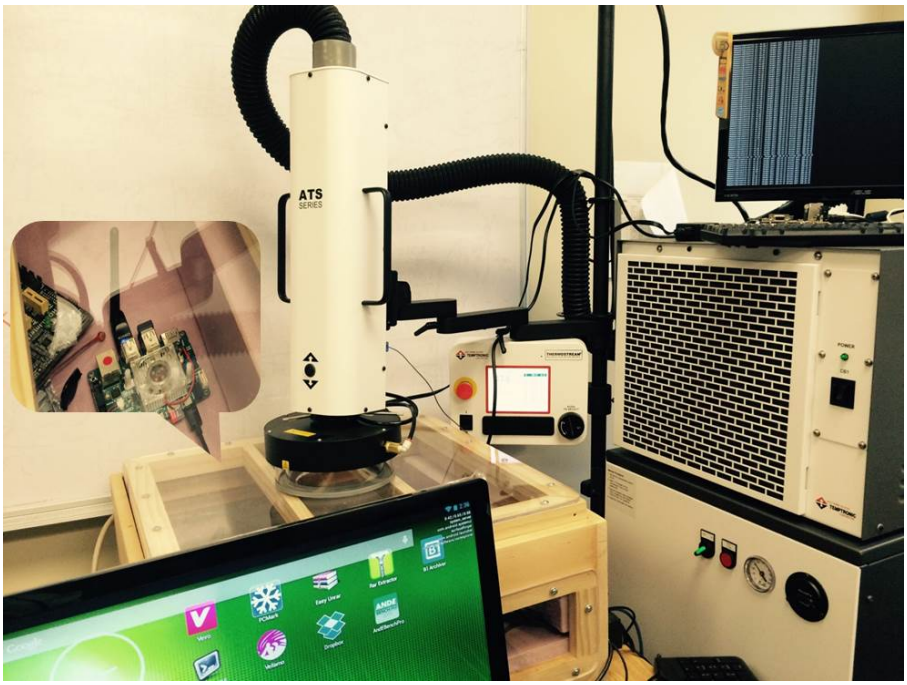


Figure 6.3: Experimental setup.

6.3 Evaluation

In this section, we first describe the experimental methodology and benchmarks. Then the parameters in the thermal prediction model are identified followed by model validation. Moreover, Fan speed control based on prediction model is first evaluated in different ambient temperature cases on the aspect of thermal control and power dissipation. After presenting the issue in high ambient environment, Falcon with refined approaches is evaluated.

6.3.1 Experimental Setup

In the experiments, we take Odroid-XU+E [46] as the reference smart gateway platform. Its OS is Android 4.2 and the detailed specification is listed in the Table 6.1. The platform provides power information of little core cluster, big core cluster, memory and GPU. The BK Precision programmable power supply [54] is connected to power up the platform, which provides a constant voltage of 5 V and a maximum current of 5 A. It records the current at a sample frequency of 4 Hz. As Figure 6.3 shows, the ThermoStream ATS air forcing system [121] is applied to control the ambient temperature and the platform board is in the sealed box under

SoC	Samsung Exynos 5410
CPU	Quad-core Cortex-A15, Quad-core Cortex-A7
Memory	2 GByte
Storage	16 GB eMMC flash
Fan	40*40*10 mm, 4000 RPM

Table 6.1: The specifications of Odroid-XU+E.

Benchmark	**Description**	**Workload**
BBench	An automatic web page rendering tool, browser benchmark.	High CPU usage
Vellamo	Benchmark suits. Multi-core bench(parsec and sysbench) are used.	multi-core CPU usage
MistBench	A set of micro benchmarks to stress each hardware component. Cache benchmark is used.	single core CPU usage
GFXBench	An OpenGL benchmark for measuring graphics performance, render quality and 3D graphics technologies.	High GPU usage
AndEBench Pro	Comprehensive embedded system benchmark, the memory and storage read/write benchmark are used.	High CPU usage, I/O

Table 6.2: The benchmarks.

the controlled environment.

We first choose the three ambient temperatures, 25°C (77°F), 15°C (59°F) and 35°C (95°F), to evaluate their influence on the thermal control so that we get a general idea. The room temperature normally is 25°C. To include outdoor situations, we run experiments in 15°C and 35°C. In addition, to pay more attention on high ambient temperature environment in which the default fan configuration works poorly, we also present the thermal control comparison in 28°C (82°F), 30°C (86°F) and 33°C (91°F) environment. The benchmarks [122, 123] used in the experiments are presented in the Table 6.2. CPU is the critical hot spot for thermal control, so the most benchmarks involve CPU test. Besides, smart gateways deal with data communication, the memory and storage system are essential. Webpage normally is the UI. Hence, we do the experiments on the five benchmarks.

6.3.2 Model Parameter Identification and Validation

As illustrated in Section 6.2, the unknown parameters are A_i for each fan speed v_i and matrix B which is the coefficient of the four components' power. The $T(k)$ and $P(k)$ are read directly from the corresponding sensors. In the implementation, the fan speed step is 10%

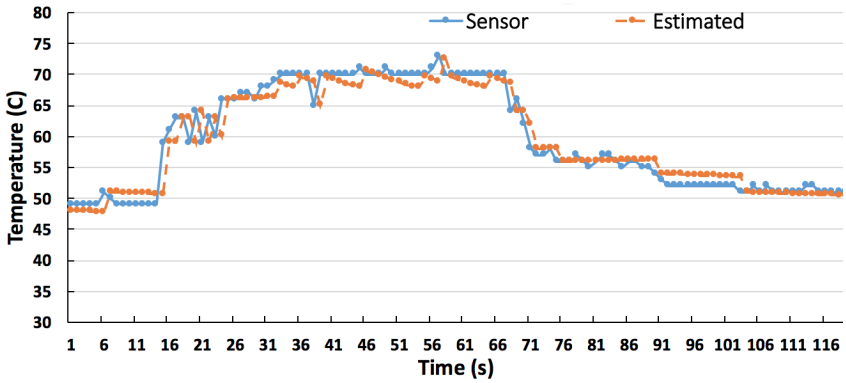


Figure 6.4: The comparison of model prediction value and sampling value.

(400 RPM) from 0% to 100%. The training benchmark is Super PI, which calculates π to a specified number of digits. To measure the influence of different power inputs, the CPU and GPU frequency are modified every thirty seconds when the application is running. The logged data and the thermal prediction model are fed to the Matlab regression tool to identify the parameters. The value of B is the average of all test runs. Figure 6.4 presents the difference of the model predicted value and the sensor logged temperature of *MistBench*, the ambient temperature is 25°C. The average prediction error of the five benchmarks is 4% of the measured value, which is acceptable.

To validate the ambient temperature estimation process, we recorded the CPU temperature in different ambient temperature cases. Figure 6.5 illustrates the temperature information in system idle state. There is a linear relationship of CPU temperature and ambient temperature. So after each idle state phrase, we can adjust current ambient temperature.

6.3.3 Thermal Control and Power Evaluation with Fan

In Falcon, one of the goals is to use fan efficiently to reduce noise and power. The decreasing of fan speed leads to noise reduction, however, it may also cause temperature increasing which influences leakage power. To analyze the total effect on the system power, we run *MistBench* with different fan speeds and record the power information as showed in Figure 6.6. Different with server case which system power mainly depends on CPU power in low fan speed situations [119], *the increase of the fan power dominates the total power on SoC platform.* The

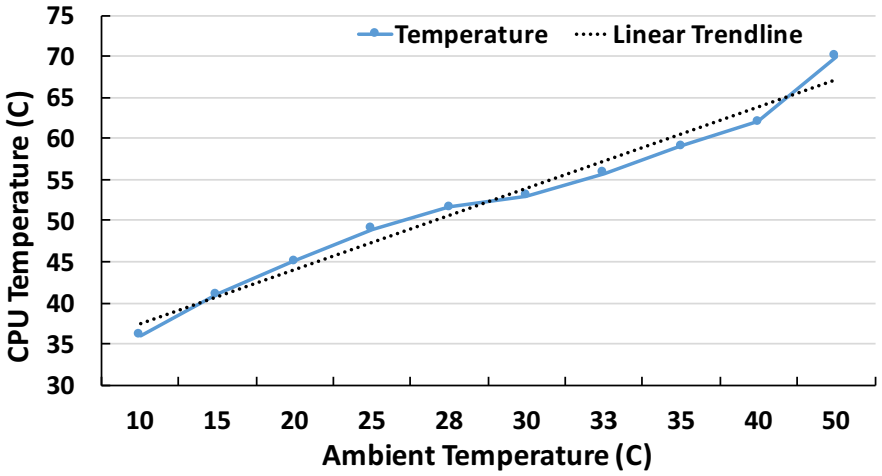


Figure 6.5: The relationship of CPU temperature and ambient temperature in idle state.

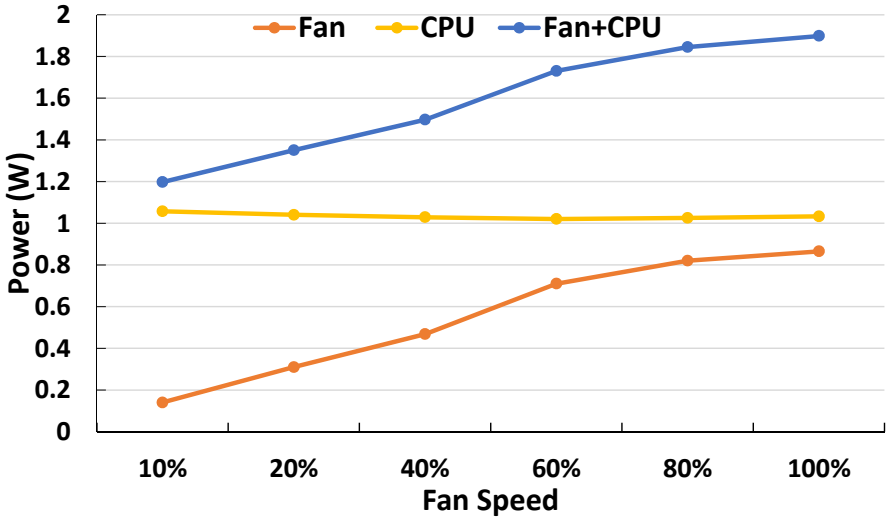


Figure 6.6: The power of the CPU and fan under each fan speed.

difference of the CPU power in 10% and 100% fan speed case is less than 0.1 W. Hence, the decreasing of fan power saves the total system power. For the leakage power, compared with the 10°C ambient temperature case , the system power increased 0.15 W which is 6% more in the 40°C case. The leakage power will be obvious if the temperature varies too much. In our experiments, the temperature difference of default fan configuration and Falcon is normally less than 10°C. Hence, the influence caused by the leakage power can be ignored.

To evaluate Falcon, we compare its thermal control ability and total system power dissipation with the platform's default fan configuration. In the default configuration, the fan runs at

Benchmark	**15°C**	**25°C**	**35°C**
BBench	4.87%	2.17%	1%
Vellamo	4.1%	5.6%	3.2%
MistBench	6.1%	7.7%	2.9%
GFXBench	2.4%	4.88%	2.27%
AndEBench Pro	5.1%	3.9%	2.08%

Table 6.3: The power savings compared with the default fan configuration under different ambient temperature.

20% speed when the CPU's temperature passes 57°C, the speed is increased to 50% and 100% if the temperature is over 63°C and 68°C respectively. So, in the experiments, the T_{ref} was set to 68°C and the fan will run at full speed when the temperature is higher.

As the gateways are deployed in different environments, the performance in three ambient temperature: 15°C, 25°C and 35°C are evaluated. Table 6.3 demonstrates the power savings compared with the default configuration for each benchmark. The power saving is the smallest in the 35°C case since the ambient temperature is high and there is no much space for fan to reduce speed. It is very easy to pass the T_{ref}. For *BBench* and *AndEBench*, their power savings in the 15°C are the maximum among the three cases. The potential reason is that their workloads are heavy and the fan is activated a lot even in low ambient temperature. The average power saving in room temperature (25°C) is 4.85%.

The power saving and noise reducing mainly come from decreasing fan speed. Next, we illustrate the influence of Falcon on the thermal side. We take *BBench* as an example, other benchmarks' result are similar. Figure 6.7 presents the fan speed information, average temperature and maximum temperature of *BBench* in different environments. The left y-axis is the percentage of the total running time that each fan speed was activated. The right y-axis is the CPU temperature. Aside from the default configuration, the *No_Fan* data was measured as the upper limit. With the ambient temperature increase, the 100% speed time increased to adjust the temperature. The major saving comes from 50% and 20% speed time of the default configuration. In our control policy Falcon, most of the time the fan is full speed or idle. When there was other fan speeds appeared (as the 25°C case), they are also less than 50% or 20%.

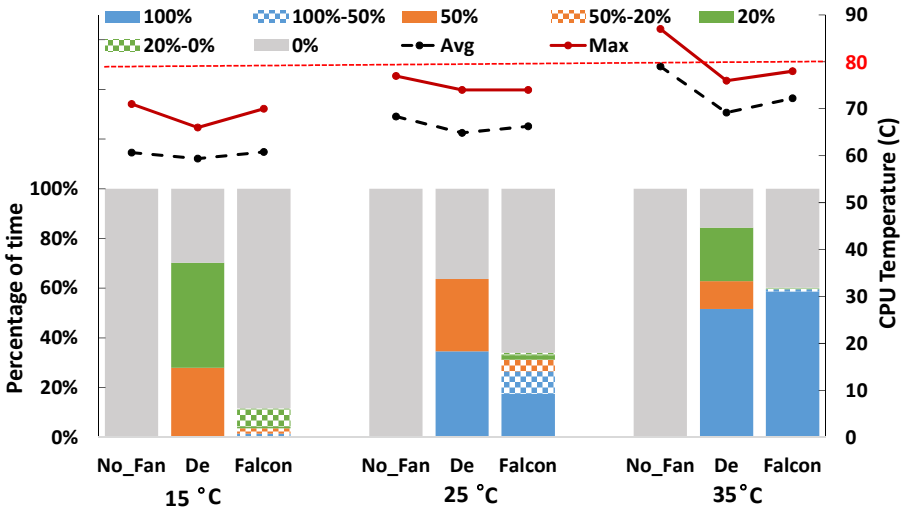


Figure 6.7: The fan speed distribution and thermal information of *BBench*. The dash line is average temperature and solid line is maximum temperature. *De* refers the default fan configuration.

The noice reduction time is about 34% on average, and the 35°C case has the least contribution. The consequence of fan speed decreasing is that the average and maximum temperature of Falcon are higher than default configuration, range from 1°C to 3°C. Assuming 80°C is the critical point, Falcon is better than the default fan configuration and can reduce noise safely within the thermal threshold.

6.3.4 Thermal Control in High Ambient Temperature Environment

As we see in the Figure 6.8, fan cannot provide the same cooling effects in different environment. The default fan configuration and prediction model based approach both work poorly in high ambient temperature case. The CPU max temperature is almost proportional to the ambient temperature. One potential reason is that the initial temperatures are not the same in different environments. To validate it, we include 35_25 case that represents the situation that the benchmark runs in 35°C environment with the initial temperature same with the 25°C case. The result shows the CPU temperature is still higher than the data in the 25°C case, so the reason that the higher initial CPU temperature leads to the cooling effect difference is excluded. The high CPU temperature results are mainly caused by the ambient temperature.

In Section 6.2, Falcon adopts two approaches to address high ambient temperature situa-

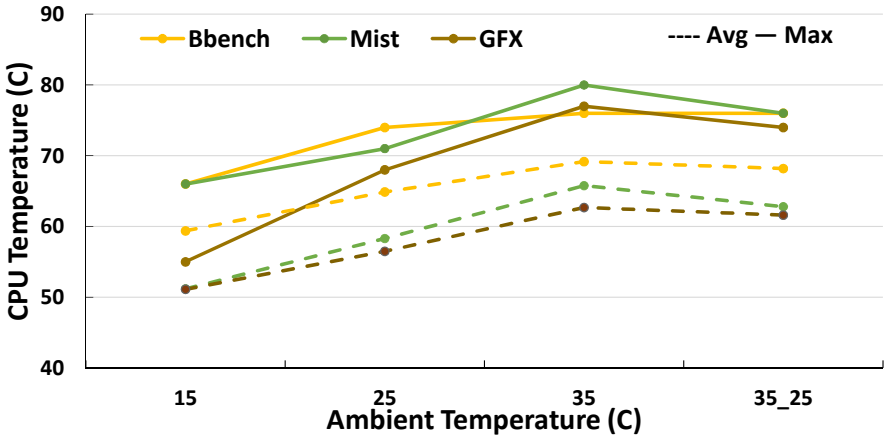


Figure 6.8: The comparison of CPU temperature in different ambient temperature cases. We show the three benchmarks to keep the figure readable. The 35_25 refers to the case that in the 35°C ambient temperature environment, the CPU temperature is cooled down to the same as the initial temperature in the 25°C case, then the benchmarks start to run.

tions. Basically, Falcon improves cooling effect by enabling fan early or restricts generated heat through adjusting CPU frequency. The enhanced fan cooling effect is preferred since it does not influence system performance. However, the approach does not always work well. Take *MistBench* and *Bbench* as examples, Figure 6.9 demonstrates their CPU temperature comparison of default fan configuration and modifying T_{ref}. Comparing with *Bbench* results, there was an obvious temperature decrease in the Mist T_ref' case, and the average CPU temperature difference is 2.37°C. For *Bbench*, the CPU temperature increased sharply at the beginning and then fluctuated around 63°C. Under the default configuration, the fan already run at 50% speed and there is no obivious heat accumulation phrase. Hence, enabling fan early is not very helpful for cooling purpose.

An important factor in the fan cooling approach is the value of the new threshold T'_{ref}. Figure 6.10 shows the *MistBench* thermal behavior under different T'_{ref} cases. We can see that the low T'_{ref} makes the temperature increase slower. While with the benchmark continues running, the temperature became steady around 56°C and 54°C. The T_ref-3 and T_ref-6 (T_ref-9 and T_ref-12) cases had similar cooling effect in the long term. So far, the T'_{ref} value is decided based on training experiments or experience, we have not found an effective way to choose

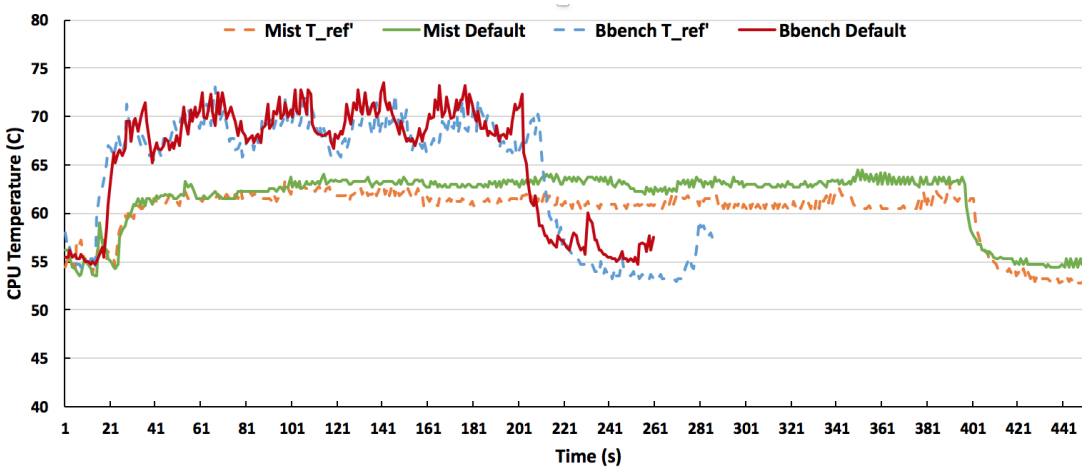


Figure 6.9: The CPU temperature comparison of default fan configuration and modifying T_{ref} for *MistBench* and *BBench* in 28°C environment. In this experiment, the T_{ref} is 8°C smaller than default configuration.

T'_{ref}.

For heavy workloads which generate a lot of heat from the beginning, Falcon maintains system under the thermal constraints through controlling CPU frequency. In our experiment platform Odroid-XU+E, there are two types of CPU cores: big core and LITTLE core. Their power consumption are very different. To generate the least amount of heat with the consider of performance loss, we chose largest frequency in LITTLE core (600 KHz) as the reduced frequency value. So in our experiments, the CPU frequency was set to 600 KHz if the temperature is over the threshold and set back if the temperature is less than the threshold.

We take *AndEBenchPro* as an example, it is a multi-thread benchmark. Figure 6.11 illustrates the CPU temperature information under the default fan configuration, modifying T_{ref} and controlling frequency approaches. As the figure shows, the CPU temperature sharply increased from 62°C to 75°C, the modifying T_{ref} approach did not improve cooling effect as the fan ran in full speed in the most time in both T'_{ref} and default cases. On the contrary, the frequency controlling approach works well. Comparing with the average temperature of 76°C in default case, the average temperature was only 61.4°C after restrict CPU frequency. However, the drawbacks of the approach is the performance loss, it took 50% more time to finish

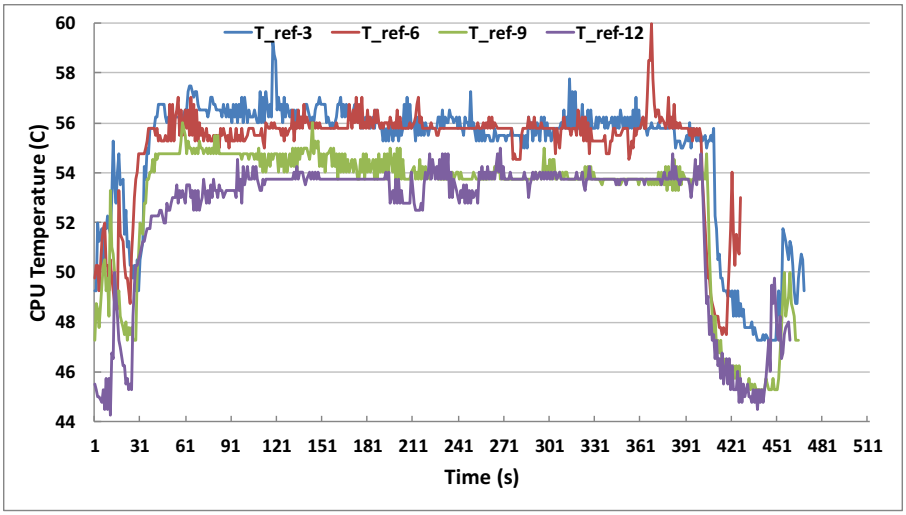


Figure 6.10: The CPU temperature comparison under different T_{ref} value for *MistBench* in 28°C environment. For example, T_ref-3 refers the threshold is 3°C smaller than default configuration.

the benchmark. Hence, when use the controlling frequency approach, we need to consider the characteristics of the workload and workload history. If heavy workload only last a few seconds, we do not need to enable control frequency approach. While if heavy workload continues running, we should sacrifice the performance to avoide burnning the device.

In addition, we also evaluated how long to stay in high (low) frequency before checking the current status. Similarly, it is a tradeoff between performance and temperature. The longer the system stay in high (low) frequency, the higher the temperature increase (decrease). The CPU temperature changes fast based on the power dissipation, so normally the temperature drops below the threshold after 1s (Falcon temperature sensor sampling time) in low CPU frequency. Hence, for simplicity, Falcon checks and modifies CPU frequency in its temperature sampling rate.

6.4 Summary

In this chapter, we presented Falcon, an ambient temperature aware thermal control policy. Compared with the default fan configuration, Falcon can reduce fan speed and save total system power with no performance loss for indoor gateways. In addition, Falcon is more adaptive in various environments. It works more effectively than the default configuration in high ambi-

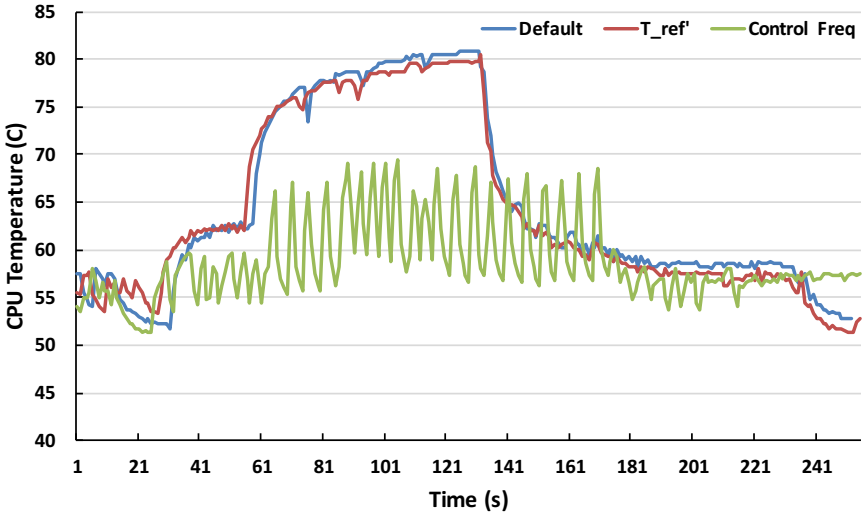


Figure 6.11: The comparison of CPU temperature of AndEBench under default configuration, modifying T_{ref} (8°C smaller) and controlling CPU frequency in 33°C environment.

ent temperature environment. To achieve that, we first built a thermal prediction model which considers the ambient temperature to predict the CPU temperature at runtime and based on that Falcon tunes the fan speed proactively under the thermal constraints. The evaluation shows that the average power saving is 4.85%. Moreover, we experimentally illustrated the influence of ambient temperature on the thermal control ability. Falcon addresses high ambient temperature issue through improving fan cooling effect and restriction power dissipation. With the modification of the reference temperature $T_{ref}{}'$, fan is enabled early to transfer heat during the CPU temperature increasing phrase. For heavy workloads, the CPU frequency is dynamically modified to control the generated heat. The average temperature can be decreased as high as 10°C with certain performance loss.

CHAPTER 7 CONCLUSIONS

Energy management is a very important issue for mobile devices. With the devices become more and more powerful, their requirement for energy also greatly increases. However, compared with the improvement of mobile hardware, the battery development is too slow. In addition, as the result of high power density, the thermal control is another issue we need to solve urgently. Hence, in this dissertation, we studied how to provide system support for energy efficient mobile computing. Basically, we analyzed main components during a general user interaction scenario, which include user behavior, the power dissipation of hardware components, operating system level scheduling and thermal control.

First, we analyzed the influence of user behavior on smartphone's energy consumption. We implemented user behavior monitoring application which logs the usage information of applications and device state (WiFi on/off, battery level, charging state, etc.). With the collected usage data and public LiveLab trace, we classified users into six types based on their application usage pattern through Fuzzy C-Means clustering algorithm. A battery prediction model is built to estimate the battery life based on user behavior and hardware usage information. The theoretical battery life and potential extended battery time for each user type, with and without hardware improvement, have been illustrated. The results show that we can double the battery time for some users if they focus on top applications and suspend rarely used ones.

Then, to figure out why slightly usage change may cause considerable energy difference, we built a power profiling tool Bugu to analyze where does the power go. Bugu analyzes power and event information for specific applications and it also contains system level information. We implemented Bugu on Android platform. It has high accuracy (95% on average) and low overhead (2.52% on average), which make it a good online power profiler. We studied 100 applications' power information using Bugu. The energy consumption of applications with same functionality varies a lot. Several implications are derived based on the observations. For examples, radio service and interrupts generated by sensors waste a lot energy since system cannot enter a deep sleep state. The power management API wakelock is abused in several

applications. The observations are useful for many energy/power related mobile researchers and developers.

Additionally, we investigated the energy saving opportunities in modern hardware design, heterogeneous platform. Heterogeneous platform is a promising trend in the near future since they can provide high performance with low power dissipation on average. We take ARM's big.LITTLE platforms Odroid-XU+E and Odroid-XU3 as experimental devices, both of them contains two types of CPU cores (high performance big core and low power LITTLE core). We compared their energy consumption with homogeneous platforms under different benchmarks which include popular mobile applications (YouTube, PhotoShop, etc.) and NAS Parallel Benchmarks (NPB). The results show that heterogeneous platforms indeed have great potential for energy saving which mostly comes from idle and low workload situations. The migration overhead is negligible. However, the scheduling is a big issue since the wrong core may cause up to 30% more energy consumption. The situation is not restricted in ARM big.LITTLE platform, it is also applied in other heterogeneous architectures including CPU-DSP, CPU-GPU, etc. Whether we can obtain the energy saving benefit from heterogeneity is mainly depends on the scheduling results and we also need to prepare for the penalty.

Finally, we studied thermal management system and developed an ambient temperature aware thermal control policy Falcon. As the devices become more and more powerful, their power dissipation also increases sharply which leads to the high temperature. The temperature problem becomes even worse due to the small size of the devices. Hence, we focus on thermal management system which is also closely related to system energy consumption. We take fan as an exemplary cooling method since it is cheap and can be installed without hardware redesign. Besides, with the blooming of Internet of Things (IoT), there is a great need for a smart gateway that can handle part of computation tasks with higher requirements on temperature as some are outdoor gateways. Hence, we take smart gateway with fan as our example platform to develop thermal control policy. Falcon is a hybrid method which leverages fan to increase heat

dissipation and DVFS to restrict heat generation. The experiments show that cooling method itself also consume a lot of energy. Hence, we built a thermal prediction model which considers components' power dissipation, temperature history and ambient temperature to estimate future temperature. Based on that, Falcon proactively tune the fan speed and it can save 4.85% total system power and reduce noise in 34% of the time on average comparing with the default fan configuration. In addition, the ability to adjust CPU frequency makes it more adaptive to high ambient temperature environment.

CHAPTER 8 FUTURE WORK

In the future, it will be beneficial to investigate how to extend and improve our approaches to further increase energy efficiency of modern mobile devices from the system level.

First, we should pay more attention to user behavior. The final goal of developing kinds of devices and saving energy is to better serve users. Hence, we can from the user side to figure out what improvement we prefer. In the dissertation, we built user behavior monitoring application and collected more than ten students' usage information. One improvement is to collect diversity users' data in a longer period, so that we may find more and finer usage patterns that provide energy saving possibilities. In addition, our current analysis process is done offline. After mining the users' data, we give suggestions that can increase battery life. As one extension, the process can be finished online if we carefully control the overhead. In this situation, it will build a record for each user and becomes more personal so that we can have better performance. Besides, the data can also be used to study the user behavior evolving process and make devices more smart.

Second, the scheduling algorithm in heterogeneous platforms should be well-studied. In the dissertation, we took ARM big.LITTLE as an example to investigate the benefits of heterogeneity. With more and more specialized mobile devices are developed, such as game hamlet, health monitoring wearable devices and so on, heterogeneous platforms will dominate the market. CPU will be released from kinds of small tasks which should be the responsibility of other processors. Hence, we need to figure out how to improve the system level energy efficiency in modern hardware architectures. Intuitively, we can compare the energy consumption of all possible scheduling options and apply the optimal one. However, it will not always be feasible due to the size of the options and the overhead. How to find good scheduling indicators that satisfy our requirements should be solved soon.

Third, the thermal control policy Falcon we proposed in the dissertation can also be applied to other cooling methods and devices with enough training experiments. Falcon is based on a thermal prediction model to tune the fan proactively. The model also works for other cooling

approaches, e.g. water cooling. Training experiments or empirical data should be needed to estimate the cooling efficiency of the cooling method. Besides, for high ambient temperature environment, new feasible frequency should be evaluated for different devices. In summary, Falcon can be used in general situations, but the value of the parameters should be re-evaluated. We can extend it to popular cooling methods and devices.

Last, we can work on the cooperation of the operating system and upper layer applications to save system energy. In the dissertation, our approaches mainly focus on how the system adjusts itself to satisfy the applications and users' requirements. From another viewpoint, the system can do better if it knows more internal things about applications rather than monitoring their behavior. Hence, it will be very helpful if we can provide some APIs to enhance the cooperation of operating system and applications. System can correctly predict future information and smoothly configure itself without wasting energy.

REFERENCES

[1] Rebecca Borison. Battery life a growing issue as more activities go mobile. http://www.mobilemarketer.com/cms/news/software-technology/17080.html, Jan 2014.

[2] Number of android applications. http://www.appbrain.com/stats/number-of-android-apps.

[3] Nathan Ingraham. Apple's app store has passed 100 billion app downloads, June 2015. http://www.theverge.com/2015/6/8/8739611/apple-wwdc-2015-stats-update.

[4] The Statistics Portal. Number of mobile app store downloads worldwide from 2011 to 2017. http://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.

[5] Microsoft. Microsoft hololens. https://www.microsoft.com/microsoft-hololens/en-us.

[6] Rob Chandhok. The future of wireless networking, 2014. Keynote.

[7] Internet of things market statistics c 2015. http://www.ironpaper.com/webintel/articles/internet-things-market-statistics-2015/.VmiPoXarSUk.

[8] NVIDIA. The benefits of multiple cpu cores in mobile devices. http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf.

[9] Nikola Pucovic. Cpu alternatives for future high-performance systems. http://web.cse.ohio-state.edu/~panda/6422/class_slides/BSC_CPU_alternatives.pdf. Barcelona Supercomputing Center.

[10] Steven Sande. A glimpse of the future: 5g wireless technology

in the labs now, May 2013. http://www.tuaw.com/2013/05/13/a-glimpse-of-the-future-5g-wireless-technology-in-the-labs-now.

[11] Daniel P. battery life to be the single main gripe of today's mobile phone user, Nov 2013. http://www.phonearena.com/news/Survey-shows-battery-life-to-be-the-single-main-gripe-of-/todays-mobile-phone-user\textunderscoreid49818.

[12] Alan Murray. What do consumers want? better batteries, not wearables. http://fortune.com/2015/01/07/what-do-consumers-want-better-batteries-not-wearables/, Jan 2015.

[13] Matthew Gast. Battery technology is not keeping pace with computing power demands. http://radar.oreilly.com/2014/03/battery-technology-is-not-keeping-pace-with-computing-/power-demands.html, Mar 2014.

[14] Greg Semeraro, Grigorios Magklis, Rajeev Balasubramonian, David H. Albonesi, Sandhya Dwarkadas, and Michael L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 29–, Washington, DC, USA, 2002. IEEE Computer Society.

[15] Yi-Chu Wang and Kwang-Ting Cheng. Energy and performance characterization of mobile heterogeneous computing. In *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, pages 312–317, Oct 2012.

[16] Chenguang Shen, Supriyo Chakraborty, Kasturi Rangan Raghavan, Haksoo Choi, and Mani B. Srivastava. Exploiting processor heterogeneity for energy efficient context inference on mobile phones. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 9:1–9:5, New York, NY, USA, 2013. ACM.

[17] ACPI. Advanced Configuration Power Interface. http://www.acpi.info/.

[18] Abhinav Pathak, Y.Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[19] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[20] Onur Sahin and Ayse K. Coskun. On the impacts of greedy thermal management in mobile devices. *Embedded Systems Letters, IEEE*, 7(2):55–58, June 2015.

[21] Qing Xie, Jaemin Kim, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '13, pages 242–247, Piscataway, NJ, USA, 2013. IEEE Press.

[22] Luca Benini. Iis-projects, thermal control of mobile devices. http://iis-projects.ee.ethz.ch/index.php/Thermal_Control_of_Mobile_Devices.

[23] Broadcom Corp. Krishna Sekar. Power and thermal challenges in mobile devices, 2013. http://www.sigmobile.org/mobicom/2013/MobiCom2013_IndustrialTalks_KrishnaSekar.pdf.

[24] Mohammad Mosharraf Ali. Yu team launches yu thermal control to tackle heating issues with yureka, yuphoria and yureka plus. http://www.yurekasupport.com/2015/09/yu-thermal-control-app-to-fix-overheating-issues.html.

[25] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh

Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM.

[26] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 329–344, New York, NY, USA, 2011. ACM.

[27] Earl Oliver. The challenges in large-scale smartphone user studies. In *Proceedings of the 2Nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, HotPlanet '10, pages 5:1–5:5, New York, NY, USA, 2010. ACM.

[28] Ahmad Rahmati, Chad Tossell, Clayton Shepard, Philip Kortum, and Lin Zhong. Exploring iphone usage: The influence of socioeconomic differences on smartphone adoption, usage and usability. In *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '12, pages 11–20, New York, NY, USA, 2012. ACM.

[29] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira, Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM.

[30] Jennifer Burge, Parthasarathy Ranganathan, and Janet L. Wiener. Cost-aware scheduling for heterogeneous enterprise machines (cash'em). In *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 481–487, Washington, DC, USA, 2007. IEEE Computer Society.

[31] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. Tarazu: Optimizing mapreduce on heterogeneous clusters. *SIGARCH Comput. Archit.*

News, 40(1):61–74, March 2012.

[32] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 77–88, New York, NY, USA, 2013. ACM.

[33] Jason Mars and Lingjia Tang. Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 619–630, New York, NY, USA, 2013. ACM.

[34] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[35] Muhammad A. Awan and Stefan M. Petters. Energy-aware partitioning of tasks onto a heterogeneous multi-core platform. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 205–214, April 2013.

[36] Jason Cong and Bo Yuan. Energy-efficient scheduling on heterogeneous multi-core architectures. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 345–350, New York, NY, USA, 2012. ACM.

[37] Rajiv Nishtala, Daniel Mossé, and Vinicius Petrucci. Energy-aware thread co-location in heterogeneous multicore processors. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 21:1–21:9, Piscataway, NJ, USA, 2013. IEEE Press.

[38] Guohui Wang, Yingen Xiong, J. Yun, and J.R. Cavallaro. Accelerating computer vision algorithms using opencl framework on the mobile gpu - a case study. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 2629–2633, May 2013.

[39] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[40] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953, March 2012.

[41] David Brooks, Robert P. Dick, Russ Joseph, and Li Shang. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *Micro, IEEE*, 27(3):49–62, May 2007.

[42] Matt Skach, Manish Arora, Chang-Hong Hsu, Qi Li, Dean Tullsen, Lingjia Tang, and Jason Mars. Thermal time shifting: Leveraging phase change materials to reduce cooling costs in warehouse-scale computers. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 439–449, New York, NY, USA, 2015. ACM.

[43] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Testore: Exploiting thermal and energy storage to cut the electricity bill for datacenter cooling. In *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pages 19–27, Oct 2012.

[44] Kai Ma Wenli Zheng and Xiaorui Wang. Exploiting thermal energy storage to reduce data center capital and operating expenses. In *Proceedings of the IEEE 19th Interna-*

tional Symposium on High-Performance Computer Architecture, pages 132–141, Feb 2014.

[45] Doug Garday and Jens Housley. Thermal storage system provides emergency data center cooling. In *White Paper Intel Information Technology*, Sep 2007.

[46] Hardkernel. Odroid xu+e. `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137463363079`.

[47] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 113–126, New York, NY, USA, 2012. ACM.

[48] Zhenqiu Huang, Bo-Lung Tsai, Jyun-Jhe Chou, Chun-Yuan Chen, Chun-Han Chen, Ching-Chi Chuang, Kwei-Jay Lin, and Chi-Sheng Shih. Context and user behavior aware intelligent home control using wukong middleware. In *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pages 302–303, June 2015.

[49] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[50] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, 2010.

[51] Russ Joseph, David Brooks, and Margaret Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In *In Workshop on Complexity Effectice Design WCED, held in conjunction with ISCA-28. Jun 2001*, June

2001.

[52] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*, ISLPED '01, pages 135–140, New York, NY, USA, 2001. ACM.

[53] Monsoon Solution Inc. Power Monitor. `http://www.msoon.com/`.

[54] B&K Precision Corp. BP Precision. `http://www.bkprecision.com/`. Model 1785B.

[55] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei 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*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.

[56] Justin Manweiler and Romit Roy Choudhury. Avoiding the rush hours: Wifi energy management via traffic isolation. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 253–266, New York, NY, USA, 2011. ACM.

[57] Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Donwon Kim, and Hojung Cha. Devscope: A nonintrusive and online power analysis tool for smartphone hardware components. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 353–362, New York, NY, USA, 2012. ACM.

[58] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.

[59] Ravi A. Giri and Anand Vanchi. Increasing data center efficiency with server power

measurements. Technical report, Intel Information Technology, 2010.

[60] International Business Machines Corporation. IBM PowerExecutive. `http://www-03.ibm.com/systems/management/director/about/director52/extensions/powerexec.html`.

[61] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[62] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 147–158. ACM Press, 2010.

[63] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Quanto: tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 323–338, Berkeley, CA, USA, 2008. USENIX Association.

[64] Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 335–348, New York, NY, USA, 2011. ACM.

[65] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 36–36, Berkeley, CA, USA, 2012. USENIX Association.

[66] Shinan Wang, Youhuizi Li, Weisong Shi, Lingjun Fan, and Abhishek Agrawal. Safari: Function-level power analysis using automatic instrumentation. In *Energy Aware Computing, 2012 International Conference on*, pages 1–6, Dec 2012.

[67] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, 2003.

[68] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.

[69] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: a platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM.

[70] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 48–63, New York, NY, USA, 1999. ACM.

[71] Narseo Vallina-Rodriguez and Jon Crowcroft. Erdos: achieving energy savings in mobile os. In *Proceedings of the sixth international workshop on MobiArch*, MobiArch '11, pages 37–42, New York, NY, USA, 2011. ACM.

[72] Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Apprehending joule thieves with cinder. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld '09, pages 49–54, New York, NY, USA, 2009. ACM.

[73] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 139–152, New York, NY, USA, 2011. ACM.

[74] Mikkel B. Kjaergaard, Jakob Langdal, Torben Godsk, and Thomas Toftkjaer. Demonstrating entracked a system for energy-efficient position tracking for mobile devices. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Ubicomp '10 Adjunct, pages 367–368, New York, NY, USA, 2010. ACM.

[75] Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[76] Ran Duan, Mingsong Bi, and Chris Gniady. Exploring memory energy optimizations in smartphones. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –8, Jul 2011.

[77] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want. Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, MobiSys '06, pages 220–232, New York, NY, USA, 2006. ACM.

[78] Eric Rozner, Vishnu Navda, Ramachandran Ramjee, and Shravan Rayanchu. Napman: Network-assisted power management for wifi devices. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 91–106, New York, NY, USA, 2010. ACM.

[79] Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu. Enabling energy efficient continuous sensing on mobile phones with littlerock. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 420–421, New York, NY, USA, 2010. ACM.

[80] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. K2: A mobile operating system for

heterogeneous coherence domains. *SIGARCH Comput. Archit. News*, 42(1):285–300, February 2014.

[81] Aaron Carroll and Gernot Heiser. Mobile multicores: Use them or waste them. *SIGOPS Oper. Syst. Rev.*, 48(1):44–48, May 2014.

[82] Aaron Carroll and Gernot Heiser. Unifying dvfs and offlining in mobile multicores. 2014.

[83] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 13–24, Washington, DC, USA, 2013. IEEE Computer Society.

[84] Ehsan Pakbaznia and Massoud Pedram. Minimizing data center cooling and server power costs. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, pages 145–150, New York, NY, USA, 2009. ACM.

[85] Raid Ayoub, Shervin Sharifi, and Tajana Simunic Rosing. Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 295–298, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[86] Zahra Abbasi, Georgios Varsamopoulos, and Sandeep K. S. Gupta. Thermal aware server provisioning and workload distribution for internet data centers. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 130–141, New York, NY, USA, 2010. ACM.

[87] Ehsan Pakbaznia, Mohammad Ghasemazar, and Massoud Pedram. Temperature-aware dynamic resource provisioning in a power-optimized datacenter. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 124–129, 3001

Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[88] R.K. Sharma, C.E. Bash, C.D. Patel, R.J. Friedrich, and J.S. Chase. Balance of power: dynamic thermal management for internet data centers. *Internet Computing, IEEE*, 9(1):42–49, Jan 2005.

[89] Hiroshi Endo, Hiroyoshi Kodama, Hiroyuki Fukuda, Toshio Sugimoto, Takashi Horie, and Masao Kondo. Cooperative control architecture of fan-less servers and fresh-air cooling in container servers for low power operation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 4:1–4:5, New York, NY, USA, 2013. ACM.

[90] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 175–186, New York, NY, USA, 2012. ACM.

[91] ARM. big.LITTLE Processing. `http://www.arm.com/products/processors/technologies/biglittleprocessing.php`.

[92] Young Geun Kim, Minyong Kim, Jae Min Kim, and Sung Woo Chung. M-dtm: Migration-based dynamic thermal management for heterogeneous mobile multi-core processors. In *DATE '15*, pages 1533–1538, San Jose, CA, USA, 2015. EDA Consortium.

[93] John Cosley. In a mobile world, not all mobile performs equally for advertisers, Apr 2014. http://marketingland.com/mobile-world-mobile-equal-78561.

[94] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.

[95] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy

trade-off for continuous mobile device location. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 285–298, New York, NY, USA, 2010. ACM.

[96] Hui Chen, Youhuizi Li, and Weisong Shi. Application-level power profiling on mobile devices: Lessons and experiences, Mar 2014. Technical Report `http://mist.cs.wayne.edu/MIST-TR-2014-001.pdf`.

[97] Android. http://developer.android.com/reference/android/app/Activity.html. Android Developer Guide, Activity.

[98] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: Measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev.*, 38(3):15–20, January 2011.

[99] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[100] Delyan Raychev, Youhuizi Li, and Weisong Shi. The seventh cell of a six-cell battery. In *third Workshop on Energy-Efficient Design, WEED*, Jun 2011.

[101] Google nexus 4 specifications. `https://support.google.com/nexus/\#topic=3415518`. Nexus Help Center.

[102] Global mobile data traffic forecast update. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[103] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[104] Green software awareness survey. `http://www.sig.eu/blobs/Nieuws/2011/Results\%20Survey-201109.pdf`.

[105] Hui Chen, Bing Luo, and Weisong Shi. Anole: A case for energy-aware mobile application design. In *Proceedings of the 1st International Workshop on Power-Aware Systems and Applications*, Pittsburgh, PA, USA, September 2012.

[106] Mian Dong and Lin Zhong. Power modeling and optimization for oled displays. *IEEE Transactions on Mobile Computing*, 11:1587–1599, 2012.

[107] Stefan Tilkov. A brief introduction to rest, Dec 2007. http://www.infoq.com/articles/rest-introduction.

[108] Pxi platform: Industry leading, pc-based platform for test, measurement, and control. National Instruments `http://www.ni.com/pxi/`.

[109] Accumulated number of application and games in the android market. http://www.androlib.com/appstats.aspx.

[110] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. Supporting distributed execution of smartphone workloads on loosely coupled heterogeneous processors. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, HotPower'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[111] Sadagopan Srinivasan, Li Zhao, Ramesh Illikkal, and Ravishankar Iyer. Efficient interaction between os and architecture in heterogeneous platforms. *SIGOPS Oper. Syst. Rev.*, 45(1):62–72, February 2011.

[112] Samsung. Heterogeneous multi-processing solution of exynos 5 octa with arm big.little technology. `http://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf`. white paper.

[113] Hardkernel. Odroid xu3, odroid xu+e. `http://www.hardkernel.com/main/main.php`.

[114] A. Gutierrez, R.G. Dreslinski, T.F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and

N. Paver. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *the proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 81–90, Austin, TX, USA, 2011.

[115] NASA Advanced Supercomputing Division. Nas parallel benchmarks. `http://www.nas.nasa.gov/publications/npb.html`.

[116] Rizwana Begum, Guru Prasad Srinivasa, David Werner, Mark Hempstead, and Geoffrey Challen. Energy-performance trade-offs on energy-constrained devices with multi-component dvfs. In *IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2015.

[117] Steven Norton. Internet of things market to reach $1.7 trillion by 2020: Idc, June 2015. http://www.marketwatch.com/story/internet-of-things-market-to-reach-17-trillion-by-2020-idc-2015-06-02-8103241.

[118] Intel. Intelligent gateways play a key role in the internet of things (iot). `http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/iot-gateway-solutions-brief.pdf`.

[119] Marina Zapater, Jose L. Ayala, José M. Moya, Kalyan Vaidyanathan, Kenny Gross, and Ayse K. Coskun. Leakage and temperature aware server control for improving energy efficiency in data centers. In *DATE '13*, pages 266–269, San Jose, CA, USA, 2013. EDA Consortium.

[120] M. Necati uzisik. Heat transfer: A basic approach. 1985.

[121] Temptronic Corporation. Thermostream ats air forcing systems. `http://www.inteststhermal.com/products/thermostream-air-forcing-systems/overview`.

[122] Anthony Gutierrez, Ronald G. Dreslinski, Thomas F. Wenisch, Trevor Mudge, Ali Saidi,

Chris Emmons, and Nigel Paver. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *IISWC*, pages 81–90, Austin, TX, USA, 2011.

[123] Jason A. Poovey, Thomas M. Conte, Markus Levy, and Shay Gal-On. A benchmark characterization of the eembc benchmark suite. *IEEE Micro*, 29(5):18–29, 2009.

ABSTRACT

SYSTEM SUPPORT FOR ENERGY EFFICIENT MOBILE COMPUTING

by

YOUHUIZI LI

May 2016

Advisor: Dr. Weisong Shi

Major: Computer Science

Degree: Doctor of Philosophy

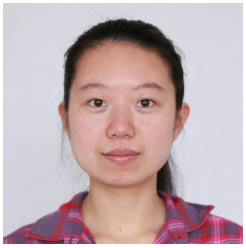
Mobile devices are developed rapidly and they have been an integrated part of our daily life. With the blooming of Internet of Things, mobile computing will become more and more important. However, the battery drain problem is a critical issue that hurts user experience. High performance devices require more power support, while the battery capacity only increases 5% per year on average. Researchers are working on kinds of energy saving approaches. For examples, hardware components provide different power state to save idle power; operating systems provide power management APIs to better control power dissipation. However, the system energy efficiency is still low that cannot reach users' expectation.

To improve energy efficiency, we studied how to provide system support for mobile computing in four different aspects. First, we focused on the influence of user behavior on system energy consumption. We monitored and analyzed users' application usages information. From the results, we built battery prediction model to estimate the battery time based on user behavior and hardware components' usage. By adjusting user behavior, we can at most double the battery time. To understand why different applications can cause such huge energy difference, we built a power profiler Bugu to figure out where does the power go. Bugu analyzes power and event information for applications, it has high accuracy and low overhead. We analyzed almost 100 mobile applications' power behavior and several implications are derived to save energy of applications and systems. In addition, to understand the energy behavior of modern hardware

architectures, we analyzed the energy consumption and performance of heterogeneous platforms and compared them with homogeneous platforms. The results show that heterogeneous platforms indeed have great potential for energy saving which mostly comes from idle and low workload situations. However, a wrong scheduling decision may cause up to 30% more energy consumption. Scheduling becomes the key point for energy efficient computing. At last, as the increased power density leads to high device temperature, we investigated the thermal management system and developed an ambient temperature aware thermal control policy Falcon. It can save 4.85% total system power and more adaptive in various environments compared with the default approach. Finally, we discussed several potential directions for future research in this field.

AUTOBIOGRAPHICAL STATEMENT

YOUHUIZI LI



Youhuizi Li is a Ph.D. candidate in the Department of Computer Science at Wayne State University. She joined the Ph.D. program in Sep 2010. She received her Masters degree in Computer Science at Wayne State University in May 2013 and received her Bachelor degree in Computer Science from Xidian University in Aug 2010. Her research interests include Energy-aware application design and optimization, Heterogeneous platform energy management and Edge computing. She has published several papers in workshops, conferences and journal, such as SUSCOM, IGCC, WEED. She has also served as a peer reviewer for journals.