EXTENDING THE BATTERY LIFE OF MOBILE DEVICE BY COMPUTATION OFFLOADING

by

HAO QIAN

B.Eng., Yunnan University, China, 2008
M.S., University of Wyoming, USA, 2011

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Abstract

The need for increased performance of mobile device directly conflicts with the desire for longer battery life. Offloading computation to resourceful servers is an effective method to reduce energy consumption and enhance performance for mobile applications. Today, most mobile devices have fast wireless link such as 4G and Wi-Fi, making computation offloading a reasonable solution to extend battery life of mobile device. Android provides mechanisms for creating mobile applications but lacks a native scheduling system for determining where code should be executed. We present Jade, a system that adds sophisticated energy-aware computation offloading capabilities to Android applications. Jade monitors device and application status and automatically decides where code should be executed. Jade dynamically adjusts offloading strategy by adapting to workload variation, communication costs, and device status. Jade minimizes the burden on developers to build applications with computation offloading ability by providing easy-to-use Jade API. Evaluation shows that Jade can effectively reduce up to 37% of average power consumption for mobile device while improving application performance.

EXTENDING THE BATTERY LIFE OF MOBILE DEVICE BY COMPUTATION OFFLOADING

by

HAO QIAN

B.Eng., Yunnan University, China, 2008
M.S., University of Wyoming, USA, 2011

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2015

Approved by:

Major Professor
Daniel Andresen

Copyright

Abstract

The need for increased performance of mobile device directly conflicts with the desire for longer battery life. Offloading computation to resourceful servers is an effective method to reduce energy consumption and enhance performance for mobile applications. Today, most mobile devices have fast wireless link such as 4G and Wi-Fi, making computation offloading a reasonable solution to extend battery life of mobile device. Android provides mechanisms for creating mobile applications but lacks a native scheduling system for determining where code should be executed. We present Jade, a system that adds sophisticated energy-aware computation offloading capabilities to Android applications. Jade monitors device and application status and automatically decides where code should be executed. Jade dynamically adjusts offloading strategy by adapting to workload variation, communication costs, and device status. Jade minimizes the burden on developers to build applications with computation offloading ability by providing easy-to-use Jade API. Evaluation shows that Jade can effectively reduce up to 37% of average power consumption for mobile device while improving application performance.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest appreciation to my advisor, Dr. Daniel Andresen for being the best source of knowledge and inspiration during my graduate experience. This dissertation would not have been possible without his constant guidance, support, and encouragement. Dan's passion on scientific research always motivates me to take the toughest research challenge. I feel lucky and honorable of having such a great experience working with him. I am looking forward to having more exciting work done together with Dan in the future.

I would like to thank everyone in my thesis committee for their participation in this whole process and for their valuable comments and suggestions.

I highly appreciate each faculty member in CIS department who has instructed and encouraged me over the past four years.

Lastly, for my fellow student friends at K-State, I enjoyed the time we spent together and we will for sure keep in touch with each other.

Chapter 1 - Introduction

1.1 Motivation

Mobile devices (smartphone and tablet) are becoming very popular nowadays. The sales of mobile devices are increasing very fast: Annual smartphone sales surpassed sales of feature phones for the first time in 2013. Worldwide sales of smartphones to end users totaled 968 million units in 2013, an increase of 42.3 percent from 2012. Sales of Android Phones passed the one billion milestone in 2014.

Mobile devices are becoming powerful and easy to take, they are replacing traditional PCs for both personal use and work purpose. Mobile devices are small computing device equipped with multiple sensors, they enable many useful applications which were unrealistic before. With the advancement of mobile hardware technology, the development of mobile applications has been changed from applications that perform basic computations to computationally intensive ones, ranging from advanced 3D games to image processing, speech recognition and augmented reality applications. Mobile applications are becoming energy hungry.

Battery technology has become one of the biggest obstacles for future growth of mobile device due to several reasons:

- Mobile device's ability to consume energy far outpaces the battery's ability to provide it, as processors are getting faster, screens are getting sharper, and devices are equipped with more sensors,. Unfortunately, technology development for batteries indicates that these limitations are here to stay, so battery life will remain the primary bottleneck for handheld mobile devices.

- Mobile devices are no longer used only for basic computation, e.g., voice communication, instead, they are used for computation intensive applications: watching videos, gaming, web surfing, and many other purposes. As a result, these applications will consume more power and shorten the battery life.

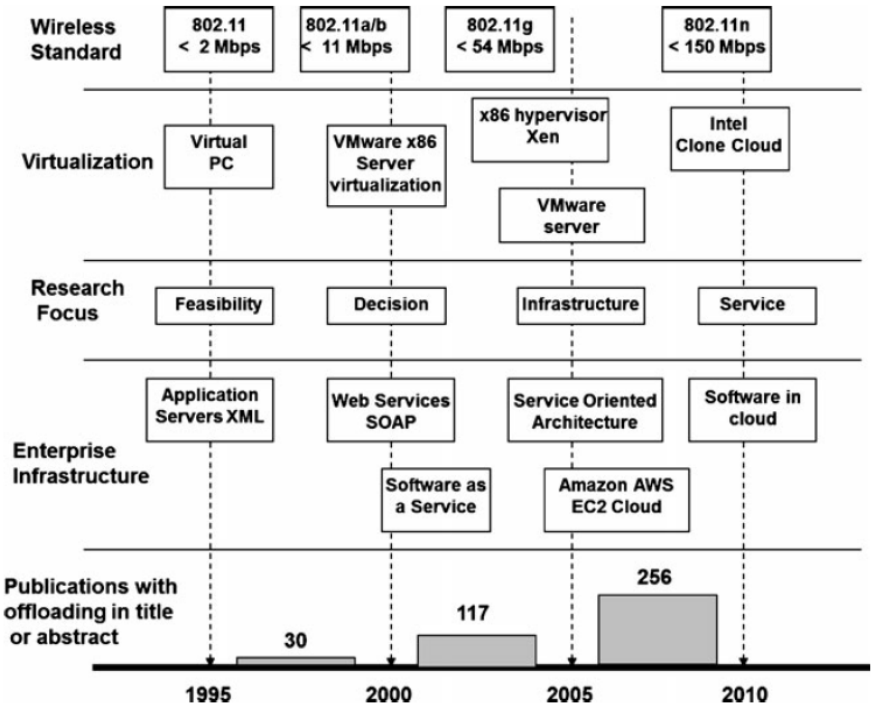
Energy is a primary constraint for mobile systems. A survey of 7,000 users across 15 countries showed that "75% of respondents said better battery life is the main feature they want". In order to provide a satisfying user experience, solving the energy problem has quickly become the mobile industry's biggest challenge.

1.2 Overview of Computation Offloading Systems

One popular technique to reduce the energy needs of mobile devices is computation offloading: mobile device can take advantage of the resource-rich infrastructure by delegating code execution to remote servers. Researchers have been studying computation offloading, focusing on different areas, e.g., making offloading decisions, developing offloading infrastructures.

Prior to 2000, researchers mostly focused on making offloading feasible [1-9]. In early 2000s, the focus moved to designing algorithms for making offloading decisions [10-24], i.e, decide whether offloading would benefit mobile users. The direction of offloading has been shifted by improvements in virtualization technology, network bandwidths, and cloud computing infrastructures [25-37]. The technology advancement has made computation offloading more practical (Figure 1-1) [38].

Figure 1-1 Enabling technologies for computation offloading



For the last two decades, in order to reduce the energy consumption of mobile device, there have been many attempts to use computation offloading. Most of these attempts took one of the following two approaches [38]:

- The first approach relies on programmers to specify how to partition a program, what computation needs to be offloaded, and how to adapt the offloading strategy to the changing network conditions. This approach is fine-grained – applications only offload the sub-parts that benefit from remote execution, thereby it leads to large energy savings. For example, a video player decodes and plays video. Offloading its decoder which is the energy-intensive part can reduce its energy consumption on the mobile device.
- The second approach is to offload full process or full VM to the infrastructure. For example, in Android, each app runs in a process which runs in a separate VM. The advantage of this approach is that it reduces the burden on programmers because they don't modify applications; instead, all their code and program state is automatically

offloaded to the remote infrastructure. The downside of this approach is that the migration cost is high because all program states need to be transferred.

Most computation offloading systems have two different goals: 1) enhance application performance and 2) reduce energy consumption for mobile device.

1.2.1 Enhance Performance

For systems target improving application performance, computation offloading is an attractive solution for shortening response time of mobile applications as applications become increasingly complex. For mobile devices which have slow processor, computation offloading is the only way to meet real-time constraints. There are many examples for such applications:

- In context-aware computing, input data stream come from different sources like GPS, maps, accelerometers, temperature sensors, etc, these data need to be analyzed together in order to obtain real-time information about users' context.
- A navigating app for a moving robot may need to recognize an obstacle before the robot collides with the obstacle. Computation may need to be offloaded if the robot doesn't have fast enough processors.

For many of these applications, the performance of resource constraint mobile systems can be enhanced by offloading computation to powerful servers.

To enhance the performance for mobile applications, we need to partition a program into two parts: 1) one part that must run on the mobile system (e.g., UI rendering and event handling for touch screen; and 2) the other part that may be offloaded. We can formulate the condition for offloading to improve performance as below. Let s_m be the speed of the mobile system, w is the amount of computation for the second part. The execution time for the second part on the mobile system is [38]

$$\frac{w}{s_m} \qquad (1)$$

To consider the execution time of the second part on the server, we ignore initial setup time of nextwork and the size of the program (server can download the program before offloading). If the second part is offloaded to a server, sending the input data d_i takes d_i/B seconds at bandwidth B. Offloading can improve application performance if and only if remote execution (including program execution and data transfer) can be performed faster than local execution. Let s_s be the speed of the server. The time to offload and execute the second part is

$$\frac{d_i}{B} + \frac{w}{s_s} \qquad (2)$$

We can see that computation offloading improve the performance of application when eq. 1 > eq.2 [38]:

$$\frac{w}{s_m} > \frac{d_i}{B} + \frac{w}{s_s} \Rightarrow w \times \left(\frac{1}{s_m} - \frac{1}{s_s}\right) > \frac{d_i}{B} \qquad (3)$$

This inequality holds for some situations:

- large w: the program contains heavy computation
- large s_s: the server is fast
- small d_i: size of data transferred is small
- large B: the bandwidth of network is high

This inequality also shows that if $\frac{w}{s_m} < \frac{d_i}{B}$, performance can't be improved by offloading, even if the server is infinitely fast (i.e., $s_s \to \infty$). Hence, when analyzing the program to be offloaded, only tasks that require heavy computation (large w) with light data exchange (small d_i) should be considered. Moreover, if we define $w\left(\frac{1}{s_m} - \frac{1}{s_s}\right) - \frac{d_i}{B}$ as the performance gain of offloading, the server's speed has diminishing return: doubling s_s will not double the gain.

1.2.2 Reduce Energy Consumption

Some computation offloading system helps reduce the energy consumption of the mobile device by delegating energy intensive computation to remote server. Similar to pervious system, when application is processed, we need to consider if a task is suitable for offloading.

The following analysis explains which task is suitable for computation offloading. Let p_m be the power of the mobile device, the energy cost of executing the task is $p_m \times \frac{w}{s_m}$ (4). Let p_c be the power of network interface of mobile device in working state, p_i is the power in idle state. So the energy consumed by network interface of the mobile device is $p_c \times \frac{d_i}{B} + p_i \times \frac{w}{s_s}$ (5).

Computation offloading saves energy when eq. 4 > eq. 5, such that [38]

$$p_m \times \frac{w}{s_m} > p_c \times \frac{d_i}{B} + p_i \times \frac{w}{s_s} \Rightarrow w \times \left(\frac{p_m}{s_m} - \frac{p_i}{s_s}\right) > p_c \times \frac{d_i}{B} \quad (7)$$

From eq. 7, we can see that tasks with heavy computation (large w) and light data size (small d_i) are suitable for computation offloading. Here, the assumption is that data is always transferred when computation offloading occurs, there are several ways to optimize this [38]:

- Data can be synchronized between mobile device and server when mobile device is charging, so data transfer doesn't hurt the battery life.
- If data resides in a remote repository (e.g., cloud), we can pass links to the server and server can download the data by itself.

1.3 Outline and Contributions

The major published contributions of this dissertation include:

- Chapter 2: Jade: An Efficient Energy-aware Computation Offloading System with Heterogeneous Network Interface Bonding for Ad-hoc Net-worked Mobile Devices

In this chapter, we proposed Jade – a computation offloading system which helps reduce energy consumption of mobile devices. We show the architecture of the system and its implementation details. Nowadays, most mobile devices are equipped with multiple wireless interface (e.g., Blue-tooth and WiFi). We provide an in-depth analysis on the energy characteristics of each interface and show how to utilize different interfaces for data transfer under various conditions in order to optimize energy usage.

- Chapter 3: Reducing Energy Consumption of Android App

  In this chapter, we show the extended version of Jade – it can offload computation to more devices, including:

  1. Android device (e.g., smartphone, tablet)
  2. Non-Android device (e.g., desktop, laptop)
  3. Cloud platform

  In order to improve the energy usage and application performance, the characteristics of servers also need to be considered (e.g., workload, battery level). We implemented a multi-level task scheduling algorithm which works on both the client and the server. The goal of the algorithm is to enhance the performance of application and balance the workload between servers. The experiments showed that such task scheduling strategy is helpful in a computation offloading system.

- Chapter 4: An Energy-saving Task Scheduler for Mobile Devices

  The size of data transferred over the network is the dominant factor affecting the performance of a computation offloading system. As the network interface of mobile device such as Wi-Fi is energy intensive, we need to reduce the amount of data transferred when computation offloading occurs. In this chapter, we propose an

innovative way to reduce the energy cost for mobile device's wireless interface: separating the task into two parts: 1) app data and 2) app code. Since most mobile devices need to be charged frequently in certain pattern (e.g., smartphone users often charge their devices at sleep time), during the charging period, data can be synchronized between client and server in advance before computation offloading happens. We show the implementation of a component (MDSS) which performs such task. With the help of MDSS, in most cases, Jade only need to offload app code to the server, thereby, we further reduced the energy consumption for mobile devices. We conducted experiments on two applications to show that MDSS is a very effective way to extend battery life.

Chapter 2 - Jade: An Efficient Energy-aware Computation Offloading System with Heterogeneous Network Interface Bonding for Ad-hoc Net-worked Mobile Devices

This chapter is published in [122] and reprinted in full in accordance with the IEEE rules.

2.1 Introduction

Mobile devices (e.g., smart phones and tablets) have become a necessity for people because they enable us to perform a wide variety of daily activities (e.g., video calls, emails, gaming, social networking, navigation, etc.). These devices typically are equipped with a relatively powerful mobile processor, a rich set of sensors and a substantial amount of memory. It allows previously unimaginable applications to be developed by integrating many sensors (e.g., motion sensors, position sensors and environmental sensors).

Battery life has become one of the biggest obstacles for future growth of mobile devices. The energy needs of mobile devices are growing fast as processors are getting faster, screens are getting bigger and sharper, and more sensors are installed. Unfortunately, the advances in battery capacity have not kept up with the growing energy needs of mobile devices.

One popular technique to simultaneously reduce the energy consumption and increase the performance of mobile devices is computation offloading: application can reduce energy consumption by delegating code execution to other devices. Traditionally, computations are offloaded to remote servers which are resource-rich.

Nowadays, it is common for people to have more than one mobile device, and they carry those devices at the same time (e.g., smart phone in the pocket and tablet in the briefcase). These devices can be networked directly (e.g., Wi-Fi and Bluetooth) without an intermediate access point. Sometimes offloading computation to these networked devices is preferable than

offloading computation to the cloud. One example is when people are using one device while the other device is charging. Offloading computation to the charging device could extend the battery life of the un-charged device. Another example is when the Internet connection is unavailable or not secure, offloading computation to locally networked devices is a good option.

In this paper, we present Jade, a computation offloading system for wireless ad-hoc networked mobile devices. Jade is targeted at mobile devices running the Android operating system. It minimizes energy consumption of applications through fine-grained code offloading while reducing the burden on application developers. Jade also reduces the energy consumption of network interfaces by dynamically choosing the most energy efficient interface (Wi-Fi/Bluetooth) for data transfer. Jade achieves these benefits by providing: 1) the runtime engine which supports computation offloading between wireless ad-hoc connected devices. By gathering the information about application and devices, the runtime engine can decide if code should run locally or remotely (offloaded); and 2) the simple programming model which helps developers to create applications that have the ability to offload computation.

We summarize our contributions here as follows:

- We present a scheduling algorithm which incorporates energy awareness and performance for a heterogeneous local cluster of devices.
- We present the design, implementation and evaluation of a complete system. Jade combines many features of the popular Android platform.
- We show that applications which support computation offloading can be implemented without heavy code modification. Android provides a natural separation between user interactive activities and background computational services. By following the similar

concept, the Jade programming model enables developers to write offloadable code with little effort.

- We evaluate the system with two applications. The result shows that computation offloading is an effective method for wireless ad-hoc networked devices. Jade can reduce up to 86% of energy consumption on mobile devices we tested, while maintaining/improving the performance of applications.

2.2 Jade Design

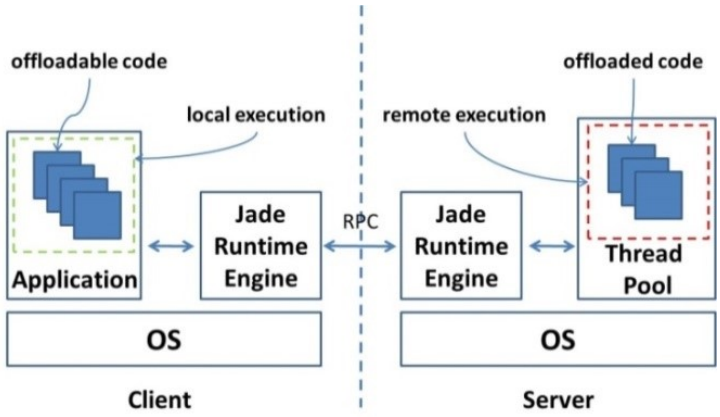
In this section, we present the high-level overview of Jade's architecture and its programming model in order to show how they all integrate into one system for distributed execution of mobile applications.

For mobile applications with heavy computation needs, computation offloading is a helpful technique to reduce the energy consumption and enhance the performance. However, it requires additional efforts and skills to develop applications with computation offloading ability, and there is no mature framework or tool for mobile application developers. We have designed Jade to minimize this effort by:

- offering the runtime engine which provides services for wireless communication, device profiling, program profiling and computation offloading. Conceptually, the Jade runtime engine automatically transforms computation on one mobile device into a distributed execution optimized for wireless connection, battery usage and capabilities of devices.
- providing the simple programming model for developing mobile applications which support computation offloading. The programming model includes a set of APIs for applications to interact with the Jade runtime engine.
- integrating with existing development tools that are familiar to developers.

In Jade, mobile applications contain offloadable code which can be offloaded to other devices (Figure 2-1). The device hosting the mobile application is called the client. The device which receives and executes the offloaded code is called the server. In Jade, if the code is executed on the client (i.e., code is not offloaded), we call it local execution. In contrast, if the code is executed on the server (i.e., code is offloaded), we call it remote execution.

Figure 2-1 Computation offloading by Jade. In Jade, applications contain offloadable code which can be executed locally or remotely. Jade runtime engine decides where to execute the code and initiates the distributed execution.



A. Jade Runtime Engine

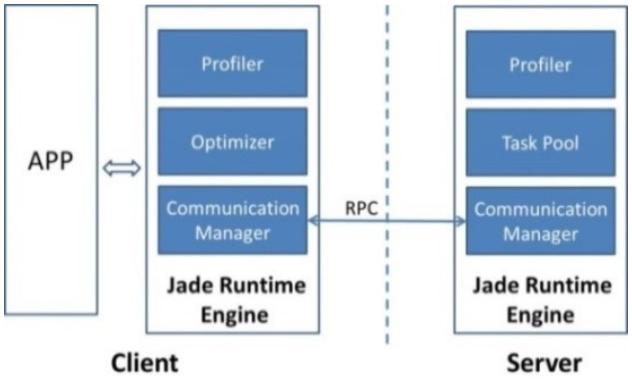
The goal of Jade is to maximize the benefits of computation offloading for mobile devices and minimize the burden on developers to develop such applications. To develop mobile applications which support computation offloading, there are several tasks we need to handle:

- Communication. In order to offload code from the client to the server, the applications should have the ability to 1) connect to other devices; 2) send data between devices; 3) coordinate with other devices for distributed execution; 4) track status of remote computation; 5) restore execution if unexpected errors happen (e.g., wireless connection lost, remote execution failure); and 6) exchange and maintain information related to connected devices (e.g. connection speed, CPU usage, battery level).

- Profiling. In order to make correct offloading decision, a profiler is needed which performs device and program profiling. Device profiling collects information about device's status, e.g., wireless connection, CPU usage, battery level. Program profiling collects information about applications, e.g., execution time, memory usage, size of data.
- Optimization. The purpose of optimization is to decide if computation is suitable for offloading, so as to maximize application's energy savings and performance.

These tasks are common for applications which support computation offloading. But it is time consuming to implement these tasks. Sometimes implementing these tasks is even harder than implementing the application itself. Jade runtime engine handles the above tasks for developers. It consists of components shown in Figure 2-2. On the mobile device, Jade runtime engine runs in the background as a group of services. Developers can utilize its various functionalities by using the easy-to-use APIs provided by the Jade programming model.

Figure 2-2 High level view of the Jade runtime engine.



In section 2.3, we describe in detail the mechanisms for the profiler, the optimizer and the communication manager, respectively.

B. Jade Programming Model

In addition to the Jade runtime engine, one of the key contributions of Jade is the programming model. The programming model is offered to developers to help build applications which support computation offloading. It is the interface between mobile applications and the Jade runtime engine. Any application developed using the programming model can interact with the Jade runtime engine and harness the power of computation offloading.

We provide the details of the Jade programming model in section 2.3.

C. Wi-Fi and Bluetooth Bonding

Today, most mobile devices are equipped with multiple network interfaces (e.g., Wi-Fi, Bluetooth and cellular interface). These interfaces have different characteristics (e.g., range, throughput and power). Each interface has its advantage and disadvantage given different applications. Ideally, we want to combine these interfaces and leverage strength of them in order to improve the application performance and reduce energy consumption.

Wi-Fi and Bluetooth are the most typical network interfaces found in today's mobile devices. Wi-Fi is known for its high throughput, long range and low energy per bit transmission cost. Bluetooth is primarily a cable-replacement technology for battery constrained mobile devices. It provides low bandwidth and covers a shorter range. The downside of Wi-Fi is the high power consumption for wake up and connection maintenance. In active state, the power of Wi-Fi is approximately 890mW, compared to only 120mW for Bluetooth. For mobile devices with limited battery capacity, Wi-Fi has been shown to account for a significant portion (up to 50%) of the total energy consumption.

The 802.11 standard defines a Power-Saving Mode (PSM), aimed at reducing the energy consumption of Wi-Fi. In PSM, the typical power consumption of Wi-Fi is effectively reduced to

250mW. In contrast, Bluetooth is optimized to be extremely low-power in idle state, typically consuming on the order of 1mW.

Given the different and often complementary characteristics of both network interfaces, it is advantageous to combine Wi-Fi and Bluetooth together, so we can utilize their strength. In Jade, we implement a component in the communication manager called Wi-Fi and Bluetooth Bonding (WBB). The job of WBB is to decide which interface (Wi-Fi/Bluetooth) will be used for data transfer according to different data transfer request of applications. The goal is to reduce the energy consumption of network interfaces.

2.3 Implementation

In this section we will highlight the important implementation details of Jade. Sections 2.3.1, 2.3.2, 2.3.3 and 2.3.4 provide the details of the key components in the Jade runtime engine. Section 2.3.5 shows the details of the Jade programming model.

2.3.1 Profiler

At runtime, before the offloadable code is invoked, the Jade optimizer determines whether the code should run locally or remotely. This decision is based on the information provided by the profiler. The profiler collects the following information: 1) the device's status (i.e., charging or not, battery level, CPU load); 2) wireless connection status, such as connected or not, the bandwidth; and 3) characteristics of the offloadable code, such as running time and size. The profiler measures the code characteristics at initialization, and it continuously monitors the device and network characteristics, because for mobile devices, these can often change (e.g., wireless connection lost, battery reaches low level). The optimizer may make wrong decision on whether the code should be offloaded based on a stale measurement. The current implementation of the Jade profiler does not implement automatic program profiling. In the rest of this section,

we provide the implementation details of Jade's techniques for device, program, and networking profiling.

1) Program Profiling

We use the DDMS debugging tool provided by the Android Development Tools (ADT) to profile applications. DDMS is a powerful tool which enables us to 1) view heap usage for a process; 2) track memory allocation of objects; and 3) profile methods of object. To measure energy consumption of applications, we use PowerTutor and Trepn Plug-in [41] for Eclipse. PowerTutor is an application for Android phones that displays the power consumed by major system components such as CPU, network interface, display, and GPS receiver and different applications. It uses a power consumption model which provides power consumption estimates within 5% of actual values. Trepn plug-in for Eclipse is a power profiling tool developed by Qualcomm for Android application developers, it is designed to allow developers to easily collect performance data from any application running on a mobile device, analyze the resulting graphs in the Eclipse IDE and modify code to build power-efficient applications.

The offloadable code is invoked multiple times, each time with a randomly chosen input. For each execution, we measure: 1) runtime duration; 2) the size of data needs to be sent to the server (i.e., the size of the code, the size of data referenced by the code and the size of data required to be returned to the client once execution is complete); and 3) energy consumption of running the offloadable code. The final result is the average of multiple invocations.

2) Device Profiling

At runtime, the profiler keeps monitoring status of the device. Android uses broadcasts to notify applications if device status changes. Applications can register broadcast listeners to receive these broadcasts. The profiler registers broadcast listeners to receive broadcasts related to

1) battery (i.e., battery level, charging or not); and 2) wireless connection (i.e., Wi-Fi turned on/off).

Due to the nature of mobile devices, the status of wireless connection could change frequently (e.g., user moves to other location). Fresh information about wireless connection is critical for the optimizer to make correct offloading decision. Similar to MAUI, we use a simple method to measure the wireless link: Each time the Jade runtime engine offloads code, the profiler measures the transfer duration to obtain a more recent average throughput. This simple approach allows the profiler to take into account both the latency and bandwidth characteristics of the network. We also build a simple energy cost model of wireless transfer using this approach: we send some synthetic data from the client to the server, varying the size of the data, and we measure the energy consumption of each transfer. This model lets us predict the energy consumption of transferring data as a function of the size of the data.

2.3.2 Optimizer

The purpose of the Jade optimizer is to pick which offloadable code to offload to the server, so as to find an offloading strategy that minimizes the client's energy consumption. The optimizer makes the offloading decision by solving an optimization problem using information collected by the profiler as input.

It requires a global view of the application and devices to decide where to execute the offloadable code. The optimizer can make adaptive decision based on the status of devices. For example: 1) if one device (client or server) is charging, the optimizer will try to offload as many computation as possible to that device in which case it is advantageous in terms of saving battery; 2) if the battery level of the server is low, the optimizer will send less code to the server (similarly if the wireless connection is bad); and 3) if the battery level of the client is low, more

code will be offloaded to the server. The goal is to reduce as much energy consumption as possible on the client without unduly burdening the battery on the server.

The characteristics of offloadable code also determine where it should be executed. The code should be offloaded only if the energy consumed to execute it locally is greater than the energy consumed to transfer it. Code which performs heavy computation on small data falls into this category. For some code, the cost of transfer outpaces the cost of local execution (e.g., light weight computation on big data), they should not be offloaded.

Based on the information provided by the profiler, the optimizer finds the best offloading strategy by solving the following problem. I represents the set of offloadable code in the application. For each offloadable code $i \in I$, E_i^e is the energy consumed to execute it locally, E_i^t is the energy consumed to transfer i between the client and the server. T_i^l is the execution time of i on the client, T_i^r is the execution time of i on the server, T_i^t is the transfer time of i. I_i is the indicator variable: $I_i = 0$ if i is executed locally, $I_i = 1$ if i is executed remotely. The optimizer needs to find the assignment for I_i such that:

- *maximizes* $\sum_{i \in I} I_i \times (E_i^e - E_i^t)$
- *guarantees* $\sum_{i \in I}(1 - I_i) \times T_i^l + I_i \times T_i^r + I_i \times T_i^t \leq l$

The first formula is the total energy savings. The second constraint stipulates that the total execution time is within *l*. *l* can be configured by developers according to different requirements.

As explained in section 2.3.5, the Jade programming model requires that every offloadable code i is independent of each other. This further simplifies the above problem. For each offloadable code i, the optimizer only needs to solve the following inequation:

$$E_i^e - E_i^t > 0$$

$$T_i^r + T_i^t - T_i^l \leq l$$

$E_i^e - E_i^t$ is the energy saving if i is executed remotely. i should be considered for offloading only when $E_i^e > E_i^t$. Similarly, the second inequation guarantees that time difference between remote execution and local execution is not greater than l.

2.3.3 Communication Manager

Computation offloading is handled by the communication manager. The communication manager is responsible for: 1) code manipulation (i.e., code transfer, code invocation); and 2) device coordination.

After the optimizer decides an offloadable code should be offloaded, the communication manager will perform the following tasks:

1. looks up the server table for available server (if no server available, then code is executed locally).
2. records information of the code in the offloaded code table, the purpose of the table is to track the status of the offloaded code.
3. offloads the code to the server.
4. the communication manager of the server receives the code, and executes it in a new thread in the task pool.
5. The communication manager of the server sends the result back to the client after the execution is complete.
6. the communication manager of the client receives the code and updates its information in the offloaded code table.

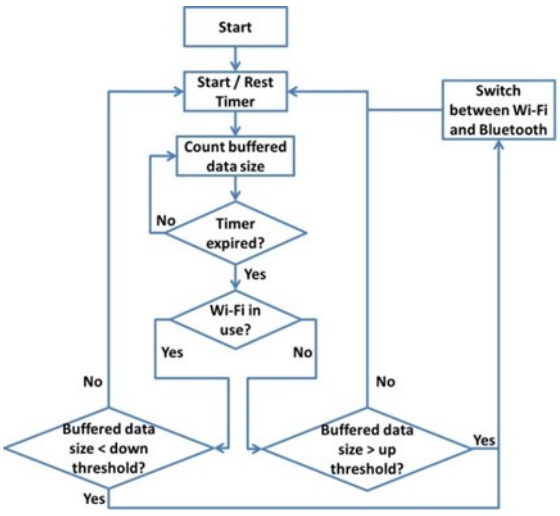
In case of remote execution failure, the communication manager has mechanisms to guarantee the correctness of the application. For example, if wireless connection is lost during

the remote execution, the communication manager of the client will re-execute the code locally, and the communication manager of the server will abandon the failed execution. If the returned code shows its result as failed, it will also be re-executed on the client as well.

2.3.4 Wi-Fi and Bluetooth Bonding

Different applications have different patterns of data transfer. For example, video streaming applications need to keep receiving a lot of data in high frequency from the server. Weather applications transfer a small amount of data with low frequency (e.g., 10 minutes, 30 minutes or 1 hour). For big data transfers (e.g., image, video), Wi-Fi is the best choice, because it provides high throughput and energy efficiency. But for small and infrequent data transfer (e.g., transfer 1KB data every 10 seconds), Wi-Fi may not be the best choice. As shown in previous section, with the implementation of PSM, the power of Wi-Fi can be reduced effectively in idle state. But Wi-Fi has high power consumption for wake up and connection maintenance. For small and infrequent data transfer, the wake up cost for Wi-Fi is high, so Bluetooth is a better choice than Wi-Fi.

Figure 2-3 Wi-Fi and Bluetooth Bonding implementation

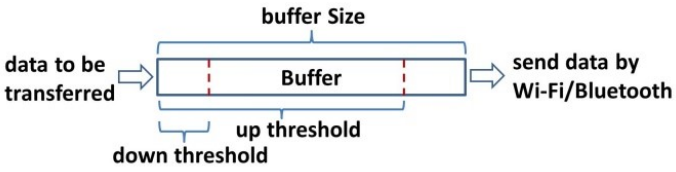


In Jade, we implement Wi-Fi and Bluetooth Bonding (WBB). WBB is adaptive to different data transfer pattern of applications. It can dynamically choose the suitable network interface (Wi-Fi/Bluetooth) for different data transfer request (Figure 2-3).

In WBB, the data to be transferred is put into a buffer. The size of the buffer is represented as S, down threshold is D and up threshold is U. The constraint is $0 \leq D < U \leq S$.

S, D and U can be configured by developers, for example, S is 500KB, D is 50KB and U is 350KB. At runtime, the buffer acts like a queue. The data to be transferred is put into the buffer from one end, then retrieved from the other end and finally sent by Wi-Fi/Bluetooth. WBB keeps monitoring the buffer and dynamically switches between Wi-Fi and Bluetooth according to the size of data in the buffer (Figure 2-4).

Figure 2-4 WBB monitors the buffer and choose suitable network interface for data transfer based on the size of data in the buffer.



This design guarantees that by properly configuring S, D and U, WBB can always make the desired choice regardless of the data production rate and the size of data. For example, if a high quality image needs to be transferred, WBB will choose Wi-Fi, because the size of the image can easily exceed the up threshold. Another example is an application which sends small data at high frequency. Due to the high data production rate, the size of data in the buffer will soon exceeds the up threshold, so Wi-Fi will be used to transfer data. In contrast, if small data is generated at low frequency, the size of data in the buffer could remain smaller than down threshold, so Bluetooth will be used for data transfer and Wi-Fi is kept in PSM.

2.3.5 Jade Programming Model

When designing applications which support code offloading, the application needs to be partitioned into sub-parts which can be offloaded to the server. There are different levels at which to partition an application (e.g., class, method, process, thread). In Jade, an application is partitioned at the class level. Developers can produce an initial partition of their applications with minimal effort by using the Jade programming model. A class simply needs to implement the RemotableTask interface if it should be considered for offloading by the Jade runtime engine.

In Jade, a class which implements the RemotableTask interface is called remotable class. An instance (object) of remotable class is called remotable object (i.e., offloadable code mentioned in previous sections). A remotable object can be executed on the client (locally) or on the server (remotely). An application developed using the Jade programming model is called Jade compatible application. At runtime, a Jade compatible application contains remotable objects which can be executed locally or remotely.

Some types of code must be executed locally, if a class contains the following code, it should not be considered for offloading:

- code that creates the user interface of the application.
- code that handles user interaction (e.g., callback method for clicking a button).
- code that access special hardware of the client which could be unavailable on the server (e.g., some smart phones are equipped with temperature sensor but some tablets are not).
- code that is not suitable for re-execution (e.g., code which perform online transaction).

Our goal of implementing the RemotableTask interface is that developers don't need to guess if a class is suitable for offloading (in terms of energy consumption and performance). Once a class doesn't contain the above code, it can implement the RemotableTask interface.

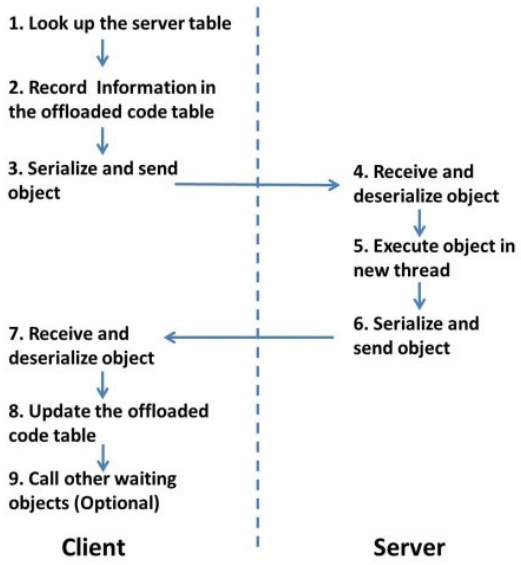
The RemotableTask interface is the key construct in the Jade programming model, it defines the following methods which will be called by the Jade runtime engine in sequence:

- preExecution, which is used to do some preparations before the remotable object performs the main task, for instance, initializing the data.
- loadData, which is used to load data from the file system of the client. If the remotable object is running on the server, invoking this method will cause the client to read data from the file system and send it to the server.
- execution, which is invoked to perform the main task.
- updateData, which is the counterpart of loadData. After the task finishes, update data in the file system. If the remotable object is running on the server, the data will be sent back to the client to update its file system.
- postExecution, which is the counterpart of preExecution. If there are any things need to be done after the task completes (e.g., disconnect with database, update log and send notification to user), we do it here.

By implementing the RemotableTask interface, the life of a remotable object can be divided into three stages: 1) before execution; 2) execution; and 3) after execution. This division is natural because it matches the steps of a general computation: 1) preparation (e.g., load data, connect to network); 2) execution, which performs computation on the data; and 3) update, which wraps up the execution (e.g., update database, notify user).

All remotable objects of an application will be considered for offloading by the Jade optimizer, if a remotable object is chosen for offloading, the Jade runtime engine will handle its remote execution following the steps shown in Figure 2-5.

Figure 2-5 Life-cycle of remotable object which is offloaded to the server.

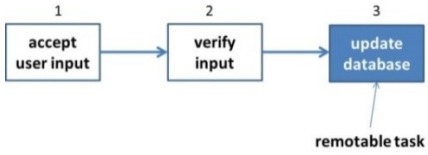


Using Jade programming model, the application development is intuitive. Figure 2-6 shows an example. Imagine we have an application which collects information entered by the user (e.g., name, phone number, address and company), verifies the information, and finally saves the information to the database on a remote server. The application could contain three steps. Step three is a good candidate for computation offloading, especially when the database operation is heavy. In the Jade programming model, we create a remotable class (UpdateInfo) for step three. The pseudocode looks like:

```
class UpdateInfo implements RemotableTask{
    preExecution(){
        open database connection;
    }
    loadData(){
        do nothing;
    }
    execution(){
```

```
        if new user
            insert user info into database;
        else
            update user info;
    }
    updateData(){
        do nothing;
    }
    postExecution(){
        close database connection;
    }
}
```

Figure 2-6 An application which contains task that could be executed remotely.



In Jade, a remotable class should perform task in-dependently. The advantage of this design includes: 1) avoid the complexity introduced by dependencies when the Jade optimizer makes the offloading decision. In fact, the algorithm of the Jade optimizer is light weight, so it consumes less energy; and 2) the application benefits from parallel execution. Since each remotable object performs independent task, they can be executed simultaneously locally or remotely, which greatly enhances the performance of the application.

2.4 Evaluation

In this section we evaluate Jade's ability to improve energy consumption and performance of mobile applications. We implemented two applications that contain heavy computation for the evaluation.

Our first application is FaceDetection. It asks user to choose some pictures from the photo gallery, then detects faces appearing in these pictures, and finally highlights them by putting a rectangle around each face. We build FaceDetection using the Jade programming model and the face detection library in Android. The code that detects faces is implemented as a remotable class, so it can be offloaded.

The second application is TextSearch, which searches text files against a library of 1000 key words. It counts how many times each key word appears in the text files. The search algorithm is implemented in a remotable class. At runtime, the text files can be searched locally or remotely.

We used an HTC One smart phone as the client. HTC one is a high end smart phone equipped with Qualcomm Snapdragon 600 quad-core 1.7GHz CPU and 2GB RAM. For the server, we use a Samsung Galaxy Tab 3 tablet and a Samsung Galaxy S4 smart phone. Galaxy Tab 3 is equipped with Intel Atom Z2560 dual-core 1.6GHz processor and 1GB RAM. Galaxy S4 has 1.9GHz quad core processor and 2GB RAM. All devices run Android 4.2.2. The client and the servers are connected by Wi-Fi and Bluetooth. We measure the energy consumption of the two applications on the client by the Trepn Plug-in for Eclipse and the PowerTutor.

At runtime, Jade can dynamically change its offloading strategy according to the battery level of devices. For example, if the battery level of the client is low and the server is charging, Jade will offload as much computation as possible to the server (aggressive mode). If the battery level of the server is low, Jade will offload less computation to the server to avoid draining its

battery (moderate mode). In the following tests, the battery level of the client is kept below 20%, and the server is charged, so Jade works in aggressive mode.

We vary the number of servers to see if it has impact on the performance of applications. For tests with one server, we use the Galaxy Tab 3 as single server. For tests with two servers, we add the Galaxy S4 as the second server. To see the impact of WBB on the energy consumption, we do each test with WBB enabled and disabled. When WBB is disabled, devices are connected with both Wi-Fi and Bluetooth, but only Wi-Fi is used for data transfer.

Figure 2-7 Energy consumption of FaceDetection.

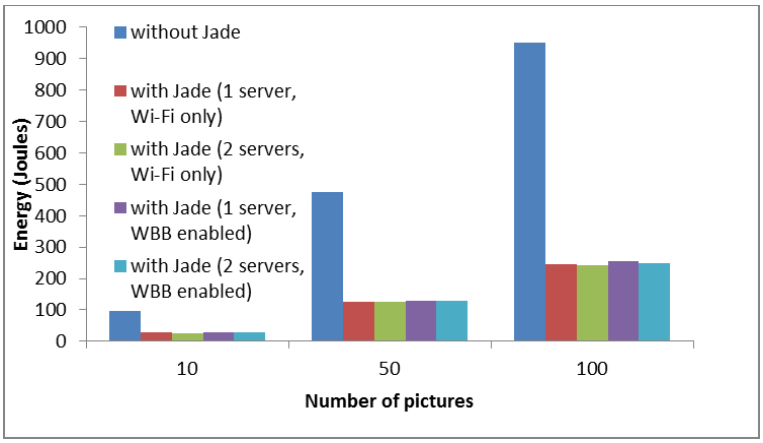


Figure 2-8 Execution time of FaceDetection.

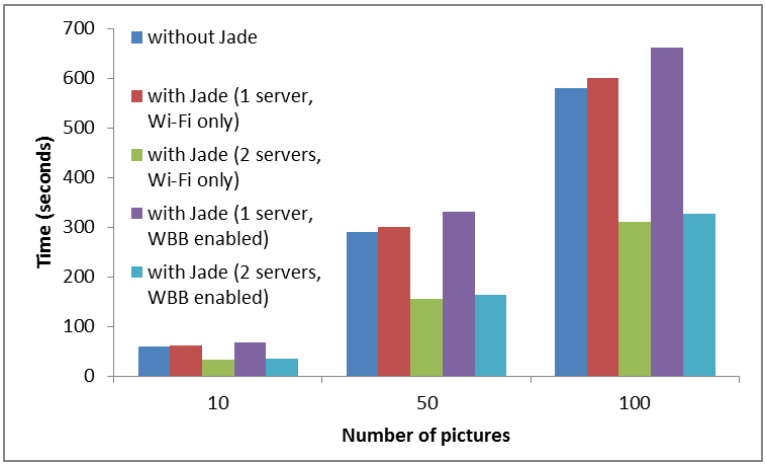


Figure 2-9 Energy consumption of TextSearch.

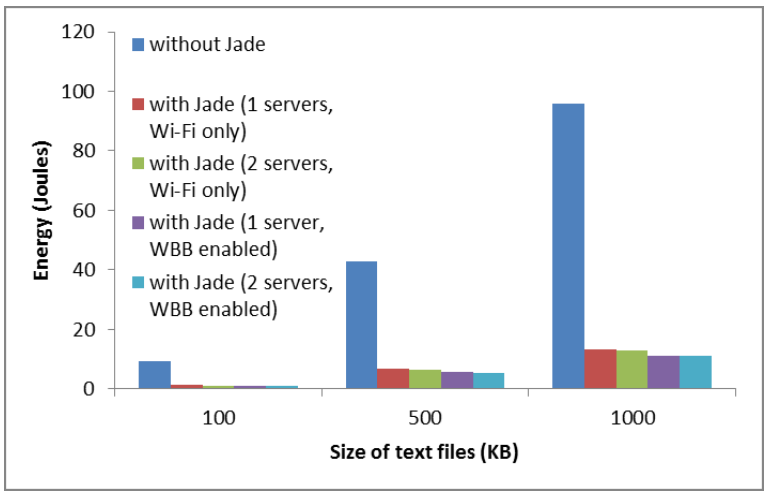
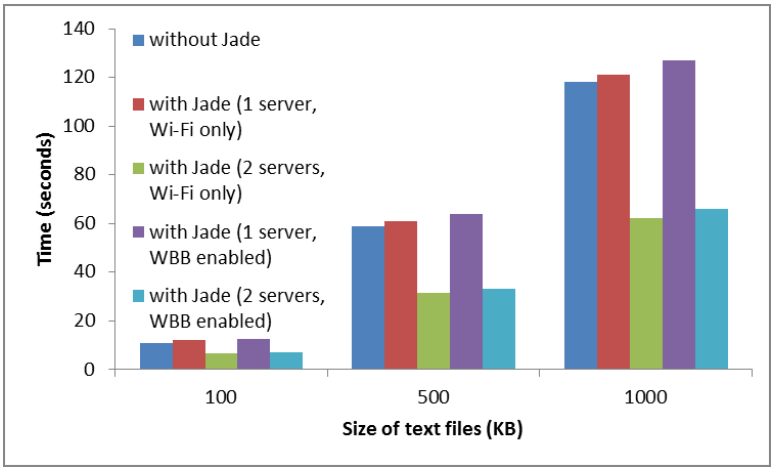


Figure 2-10 Execution time of TextSearch.



To evaluate how much energy Jade saves for FaceDetection, we execute it on the client with Jade enabled and disabled, and varying the number of pictures to detect. We only use pictures smaller than 200KB. The results are shown in Figure 2-7 and Figure 2-8.

We use similar method to evaluate TextSearch: execute it with Jade enabled and disabled, and varying the total size of text files that are searched. Each text file is 50KB. The energy consumption and execution time are shown in Figure 2-9 and 2-10.

From the tests, we can see that Jade saves 74% of energy for FaceDetection and 86% of energy for TextSearch. For execution time, when there is only one server, execution without Jade is a little faster than execution with Jade: there is 3.7% and 3% slowdown for FaceDetection and

TextSearch. This is due to two reasons: 1) the server is less powerful than the client (CPU and memory); and 2) transfer overhead. When there are two servers, the performance improves: Jade reduces 43% and 48% of execution time for FaceDetection and TextSearch.

For FaceDetection, WBB does not reduce energy consumption compared with Wi-Fi only mode. This is because each remotable object contains an image, its size exceeds the up threshold of the buffer, so Bluetooth is never used for data transfer. For TextSearch, the application send small data (50KB per file) at long interval, so WBB effectively reduced 10% more energy compared with Wi-Fi only mode.

WBB keeps monitoring the buffer, and data needs to be stored in the buffer before transfer. This introduces overhead, so for both applications, the execution time with WBB is longer than Wi-Fi only mode, the execution time increased at most 10%.

The results demonstrate that Jade can effectively reduce battery consumption of applications while improving the performance. It also shows that WBB can dynamically choose suitable network interfaces. For applications which send small piece of data at low frequency, WBB can further reduce the energy consumption for applications.

2.5 Related Work

Mobile devices have limited resources such as battery capacity, storage and processor performance. Computation offloading is an effective method to alleviate these restrictions by sending heavy computations to resourceful servers and receiving the results from these servers. Many issues related to computation offloading have been investigated in the past decade: making offloading feasible, making offloading decisions, and developing offloading infrastructures.

Jade is built upon previous research done in program partitioning, code offloading, and remote execution. In this section, we give an overview of what has been proposed by these researches and how they relate to Jade.

There are many earlier efforts on code migration, including systems for mobile applications [53 -57]. The primarily focus of these systems is on enabling code and data to easily move between devices in a distributed system. None of these systems attempted to ease the development process for the programmer, nor did they focus on reducing energy consumption for mobile device.

Cuervo et al. proposed MAUI [39], a system that enables energy-aware offloading of mobile code to the infrastructure. MAUI enables developers to produce an initial partitioning of their applications by annotating methods and/or classes as remotable. At runtime, MAUI solver decides which remotable methods should execute locally and which should execute remotely. Unlike MAUI, Jade provides the programming model which enables developers to create remotable object. This has one significant benefit: the profiling and optimization overhead is low. Because In Jade, each remotable object is an independent unit performing some tasks, for the optimizer, it only needs to decide if a remotable object should be offloaded regardless of the other code of the application.

Chun et al. proposed CloneCloud [40], an application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. In CloneCloud, to offload computation, threads need to be paused, all states of the threads need to be transferred to the server, and finally threads resume on the server. The offloading is expensive, especially when the client and the server are both resource

constraint mobile devices. In contrast, code offloading in Jade is lightweight. Remotable objects are serialized, transferred and deserialized, the overhead is much lower than thread migration.

2.6 Future Work

In our future work we will improve the Jade profiler by providing automatic application profiling. This will further reduce the burden on application developers. We will also study the impact of automatic profiling on the effectiveness of Jade, since it introduces more overhead.

2.7 Conclusion

In this paper, we have presented Jade, a system which enables computation offloading for wireless ad-hoc networked mobile devices. Jade can effectively reduce the energy consumption of mobile devices (over 75% in our examples), and dynamically change its offloading strategy according to the status of devices.

We have evaluated Jade with two applications, a face detection application and a text search application. The result shows that Jade can reduce the energy consumption effectively for both applications while maintaining/improving the performance.

Chapter 3 - Reducing Energy Consumption of Android App

3.1 Introduction

Mobile devices, such as smart phones and tablets, have become a necessity because they allow people to perform a wide variety of useful activities (e.g., video calls, emails, gaming, social networking, navigation, etc.) with mobility. These devices typically are equipped with a relatively powerful mobile processor, a rich set of sensors, and a substantial amount of memory. It allows previously unimaginable applications to be developed by integrating sensors such as motion sensors, position sensors, and environmental sensors.

However, battery life has become one of the biggest ob-stacles for mobile device advancements. Performance de-manded by smartphones and tablets is increasing at a much faster rate than technological improvements in battery capacity. The need for increased performance of mobile devices directly conflicts with the desire for longer battery life.

One popular technique to reduce energy consumption of mobile devices is computation offloading in which an application reduces energy consumption by delegating code execution to other devices. Traditionally, computations are offloaded to remote, resource-rich servers. Selection of a proper offloading strategy can reduce power consumption and simultaneously enhance mobile device performance.

In this paper, we present Jade, an energy-aware computation offloading system for mobile devices [45 - 51]. Jade, built for mobile devices running Android operating system, minimizes energy consumption of mobile applications through fine-grained computation offloading by providing the following benefits:

1. A runtime engine that enables computation offloading from mobile device to servers. By monitoring application and device status, the runtime engine automatically decides if code should run locally or remotely.
2. A programming model that helps developers create applications with computation offloading ability.

Our contributions are summarized as follows:

- We present the design and implementation of a complete system. Jade is able to offload computation from an Android device to any Android/non-Android devices.
- We present a multi-level scheduling algorithm that automatically and seamlessly transports workloads to the appropriate server based on performance and energy needs.
- We evaluate Jade with two applications. Results indicated that Jade can effectively reduce up to 39% of average power consumption for mobile device, while reducing the execution time of application.

3.2 System Architecture

In this section, we present the high-level design of Jade and its programming model in order to demonstrate how they integrate into one system, thereby supporting distributed execution of mobile applications.

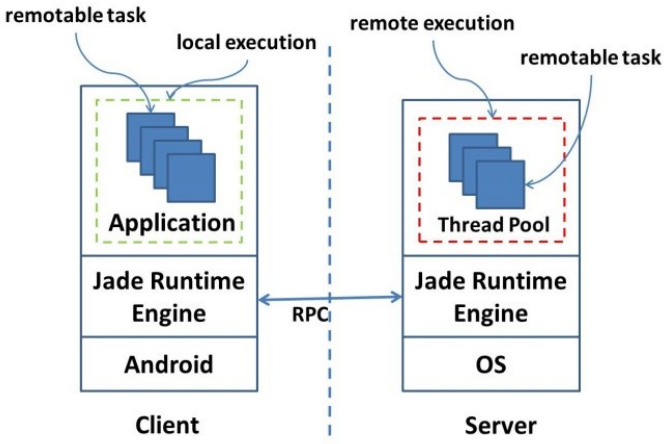
For mobile applications with heavy computation needs, computation offloading is an effective method to reduce energy consumption and enhance performance. However, it requires additional efforts and skills to develop applications with computation offloading ability and, unfortunately, no mature frameworks or tools exist for mobile application developers. We

designed Jade to minimize the workload of developing applications with computation offloading ability by:

- Offering the Jade runtime engine which provides services for wireless communication, device profiling, program profiling and computation offloading. Conceptually, the Jade runtime engine automatically transforms application execution on one mobile device into a distributed execution optimized for wireless connection, power usage, and server capabilities.
- Providing an easy-to-use programming model for developers to build mobile applications that support energy-aware computation offloading.

In order to increase understanding of this offloading system, terms used in Jade must be defined. The mobile device that offloads computation is called the client. The device that receives and executes the offloaded code is called the server. Mobile applications contain remotable tasks which can be offloaded to the server (Figure 3-1). If a remotable task is executed on the client (i.e., it is not offloaded), we call it local execution. In contrast, if a remotable task is executed on the server (i.e., it is offloaded), we call it remote execution.

Figure 3-1 Jade enables computation offloading for mobile applications. Ap-plications contain remotable tasks that can be offloaded from the client to the server. Jade runtime engine automatically decides where to execute remotable tasks and initiates distributed execution.



3.2.1 Jade Runtime Engine

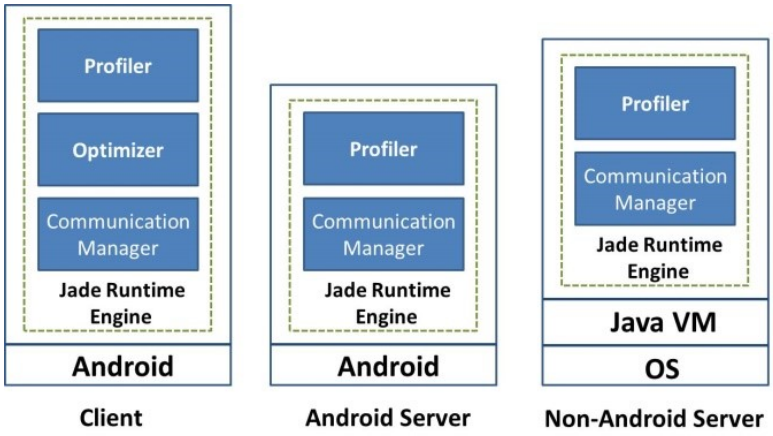
In order to support computation offloading for applica-tions, a computation offloading system is needed to handle some essential tasks such as:

- Communication. In order to offload code from the client to the server, the system should be able to 1) connect to other devices; 2) send data between devices; 3) coordinate with other devices for distributed execution; 4) track status of remote execution; 5) restore execution if unexpected errors occur (e.g., wireless connection lost, remote execution failure); and 6) exchange and record information related to all connected devices (e.g., connection speed, CPU usage, battery level, hardware configuration).
- Profiling. In order to make correct offloading decisions, the system should have fresh information regarding status of the device and application. Device profiling is the process of collecting information pertaining to device status, such as wireless connection, CPU usage, and battery level. Similarly, program profiling collects information about applications, such as execution time, energy consumption, memory usage, and data size.
- Optimization. The system should be able to deter-mine an optimized offloading strategy in order to maximize application's energy savings and performance.

The goal of Jade is to maximize the benefits of energy-aware computation offloading for mobile applications, at the same time, minimizing the burden on developers to build such an application. By proving the Jade runtime engine, computation offloading is handled automatically in the background, allowing application developers to focus on the application without implementing the computation offloading mechanism.

Jade runtime engine components are shown in Figure 3-2. On an Android device, the Jade runtime engine runs as a group of background services. Jade supports two types of server: 1) Android server is device running Android and 2) Non-Android server is device running operating systems such as Windows and Linux. However, non-Android servers must have Java VM in order to support Jade, because Jade runtime engine runs as a Java program on a non-Android server.

Figure 3-2 High-level design of the Jade runtime engine. Jade runtime engine supports Android/non-Android device.



Section 3.4 details the components of the Jade runtime engine.

3.2.2 Jade Programming Model

In addition to the Jade runtime engine, another key contribution of our system is the Jade programming model which helps developers efficiently build applications with computation offloading ability. Jade programming model provides APIs in order for mobile applications to interact with the Jade runtime engine. Use of the programming model allows developers to efficiently build applications that harness the power of computation offloading.

Details of the Jade programming model are provided in Section 3.4.

3.3 Multi-level Task Scheduling

Jade decides where to execute remotable tasks dynamically generated in an application (i.e., local execution or remote execution). Based on dynamic run time behavior of remotable tasks, such as execution time and energy consumption, remotable tasks should be offloaded to appropriate servers.

We implement a multi-level task scheduling algorithm in Jade that enables energy and performance-aware task scheduling. The algorithm 1) incorporates server status (e.g., type of power supply, computing capability, connection speed); 2) balances the workload between servers; and 3) offloads tasks to the most appropriate server according to the energy and computing demand of the task.

The multi-level task scheduling algorithm runs on the client and the server. Details are discussed in Sections 3.3.1 and 3.3.2.

3.3.1 Task Scheduling on the Client

The goals of task scheduling on the client include determining the appropriate server for each remotable task and balancing the workload for servers.

Tasks have varying computation demands. Tasks with heavy computation consume more energy, thereby requiring additional execution time. Servers also have varying characteristics. For example, mobile devices have less powerful hardware, such as CPU and RAM, and limited power supplies. Compared to mobile devices, desktop and laptop computers are often equipped with more powerful hardware and unlimited power supplies, making them better choices for tasks with heavy computation.

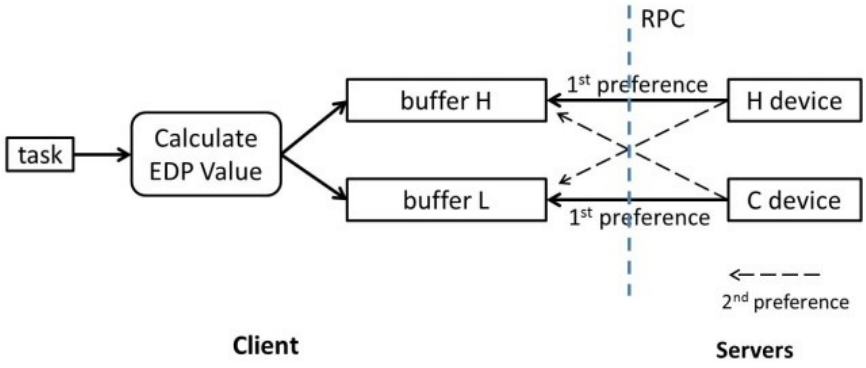
Server workload must also be considered. If a server currently operates with a high workload, it should not be a site to which new tasks are offloaded. In contrast, if a server is idle or has a light workload, it is able to receive new tasks.

We used Energy Delay Product (EDP) to measure computation demand of a remotable task. EDP, a performance measure that considers power and execution time, is defined as $EDP = T \times E = T^2 \times P$. For a remotable task *i*, T is execution time of *i*, E is energy cost to execute *i*, and P is average power consumption to execute *i*. Tasks with high computation demand should have longer execution time and consume more energy than tasks with low computation demand. Therefore, EDP can be used to classify tasks based on computation demand.

A distributed computing environment contains many useful information concerning execution parameters and performance that is readily available at each server but not readily available to the client. As a result, an information gap exists between the client and servers. In order to balance server workloads, the client ideally would have extensive information exchange with servers. Server status can change quickly, so the client must frequently gather information, thereby incurring many data transfers that consume energy. To reduce unnecessary energy costs, we used work stealing as a task scheduling strategy. Work stealing is a special case of task migration in which a "starving" device attempts to steal tasks from a "loaded" device [43]. In Jade, the servers attempt to steal tasks from the client. The server monitors its own status and requests new tasks based on level of its current workload. For example, if the server was already overloaded, it would not ask for new tasks until remaining tasks were completed. The client does not need to monitor server status when using work stealing, so the amount of data transferred between devices is reduced, consequently saving energy.

On the client, two distinct buffers are used to store remotable tasks before offloading. One buffer is used for tasks with heavy computation (buffer H) and the other buffer is used for tasks with light computation (buffer L). When a remotable task i is generated, Jade calculates the EDP value of i, and based on its EDP value, i is put into buffer H or L. We also run benchmarking applications on every server to classify server as High Performance Device (H device) or Performance Constraint Device (C device). H devices steal tasks from buffer H and only steal from buffer L when buffer H is empty. Similarly, C devices steal tasks from buffer L and only steal tasks from buffer H when buffer L is empty. If only one kind of device is present (e.g., all servers are C devices), they steal tasks from both buffers (Figure 3-3).

Figure 3-3 Servers stealing tasks from buffers of the client



3.3.2 Task Scheduling on the Server

Because mobile device users often spend limited time on one application, response time is very important for mobile applications. Given many similar applications to choose from, users do not use an application which feels sluggish, hangs, or freezes. To increase application responsiveness during computation offloading, the servers must return results of remote task execution as quickly as possible.

In the server, we used Highest Response Ratio Next (HRRN) algorithm to schedule tasks. In HRRN, the priority of each job depends on its estimated execution time and the amount of

time the job has spent waiting. Jobs increase in priority as they wait, thereby preventing indefinite waiting (starvation). HRRN favors shortest jobs, but it also limits waiting time for long jobs. The priority of a task is defined as [44]:

$$priority = \frac{waiting\ time + estimated\ run\ time}{estimated\ run\ time}$$

By selecting shorter tasks to execute, the client receives results frequently, making the application more responsive to the user.

3.4 Implementation

This section highlights important implementation details of Jade. Sections 3.4.1, 3.4.2, and 3.4.3 describe components in the Jade runtime engine. Section 3.4.4 shows details of the Jade programming model.

3.4.1 Profiler

When a remotable task is generated at run time, the Jade optimizer determines whether it should be executed locally or remotely. This decision is based on information provided by the profiler. The profiler collects the following information: 1) device status, such as charging status, battery level, CPU load, and wireless connection status); and 2) characteristics of remotable tasks, such as execution time, size, and energy consumption. The profiler measures characteristics of a remotable task during task's first execution and continuously monitors the status of devices because mobile device status changes frequently (e.g., loss of Wi-Fi connection, reaching low battery level). The optimizer may make wrong decision on whether the code should be offloaded based on a stale measurement.

Offline and online methods are available to measure power consumption of mobile devices and applications. An offline method, often used under laboratory conditions, uses external measurement tools. This method generates accurate results but requires special skills

and equipment. However, most developers and users cannot realistically perform offline power measurement. The online method overcomes limitations of the offline approach by utilizing the Battery Monitoring Unit (BMU), which provides information related to battery (e.g., supply voltage, current, and battery temperature). A majority of current mobile devices are equipped with BMU. However, online method may not be as accurate as offline method because the information update rate of BMU is much lower than external measurement tools. According to previous research of mobile device power modeling, online method can generate satisfying results. The Jade profiler uses the online approach to measure energy consumption for devices and applications. The evaluation showed that Jade can make correct offloading decision when using information generated by this approach.

3.4.1.1 Program Profiling

When a new remotable task is generated, a unique name is assigned to it. (We combine the package name of the application and class name of the remotable task to form a unique name.) During the remotable task's first execution, the profiler collects the task's energy consumption E, execution time T, and size S. Then the information is provided to the optimizer. Based on the cost model, the optimizer decides if the task is suitable for offloading. Finally, all information concerning the remotable task is recorded in a database. When the same or similar task is executed again in the future, Jade will find its information in the database in order to understand whether or not the task should be offloaded.

With our automatic program profiling, developers only need to identify potential remotable tasks in a program and mark them according to the Jade programming model. When an application is installed on a mobile device, developers only need to run the application, invoking as many remotable tasks as possible. Jade automatically analyzes marked tasks in the

background and decides whether or not they are remotable. Automatic program profiling greatly reduces the amount of work required to perform program power analysis, thus speeding up application development.

3.4.1.2 Device Profiling

At runtime, the profiler also continuously monitors device status. Collected information includes battery (i.e., battery level, charging status), wireless connection (i.e., Wi-Fi turned on/off, throughput), and CPU load.

Due to the nature of mobile devices, wireless connection could change frequently because a user could change locations. Fresh information about wireless connection is crucial in order for the optimizer to make correct offloading decisions. Similar to MAUI, we used a simple method to measure the wireless link: Each time Jade offloads code, the profiler measures the transfer duration in order to obtain more recent average throughput. This simple approach takes into account both the latency and bandwidth characteristics of the network. We also built a simple energy cost model for wireless transfer using this approach: We send synthetic data from the client to the server, varying the size of the data, and we measure energy consumption of each transfer. This model allows us to predict energy consumption of data transfer as a function of data size.

3.4.2 Optimizer

The purpose of the Jade optimizer is to choose suitable remotable tasks to offload to the server in order to find an offloading strategy that minimizes the application's energy consumption. The optimizer makes the offloading decision by solving an optimization problem using information provided by the profiler.

Characteristics of a remotable task determine where it should be executed. The code should be offloaded only if the energy cost to locally execute it is greater than the energy cost required to transfer it. For example, code performing heavy computation on small data is suitable for offloading. For some code, the energy cost of transfer outpaces the energy cost of local execution (e.g., lightweight computation on big data), such code should not be offloaded.

Based on information provided by the profiler, the optimizer determines the best offloading strategy by solving the following cost model. I represents the set of remotable tasks in the application. For each remotable task $i \in I$, E_i^e is energy consumed to execute i locally. E_i^t is energy consumed to transfer i between the client and the server. T_i^l is the execution time of i on the client. T_i^r is the execution time of i on the server. T_i^t is the transfer time of i. I_i is the indicator variable: $I_i = 0$ if i is executed locally, $I_i = 1$ if i is executed remotely. The optimizer must find the assignment for I_i such that:

- *maximizes* $\sum_{i \in I} I_i \times (E_i^e - E_i^t)$
- *guarantees* $\sum_{i \in I} (1 - I_i) \times T_i^l + I_i \times T_i^r + I_i \times T_i^t \leq l$

The first formula is total energy savings. The second constraint stipulates that the total execution time is within l. Developers can configure l according to their specific requirements.

As explained in Section 3.4.4, the Jade programming model requires that every remotable task i to perform independent job, so no dependencies exist between tasks. This requirement further simplifies the cost model. For each remotable task i, the optimizer must solve only the following inequation:

$$E_i^e - E_i^t > 0$$
$$T_i^r + T_i^t - T_i^l \leq l$$

$E_i^e - E_i^t$ is the energy saving if i is executed remotely. i should be considered for offloading only when $E_i^e > E_i^t$. Similarly, the second inequation guarantees that time difference between remote execution and local execution is not greater than l.

3.4.3 Communication Manager

Code offloading is handled by the communication manag-er. The communication manager is responsible for code manipulation (i.e., code transfer, code execution) and device coordination.

The communication manager handles code offloading by following these steps:

1. Records code information in the offloaded code table (Table 1). The purpose of the table is to track the status of offloaded code.
2. Offloads the code to the server.
3. The server receives the code and executes it in a new thread.
4. The server sends the code back to the client after execution is complete.
5. The client receives the code and updates code information in the offloaded code table.

Table 3-1 Example of offloaded code table

ID	Offloaded	Server	Returned	Result
0001	true	192.168.49.1	true	finish
0002	true	192.168.49.1	false	
0003	true	192.168.49.1	true	fail

For remote execution failure, the communication manager has mechanisms to guarantee correctness of the application. For example, if wireless connection is lost during remote execution, the client's communication manager re-executes the code locally and the server's communication manager abandons the execution. If the returned code shows its result as failed, it is also re-executed on the client.

3.4.4 Jade Programming Model

When designing applications that support code offloading, the application must be partitioned into sub-parts which can be offloaded to the server. Applications can be partitioned at various levels, such as class, method, process, and thread. An application is partitioned at the class level in Jade. Developers can produce an initial partition of their applications with minimal effort using the Jade programming model.

To be considered for offloading, a class must implement one of two interfaces: RemotableTask interface or RemotableGenTask interface. As mentioned, Jade supports Android and non-Android servers. If a class contains Android code, it must run on Android devices, so the class must implement RemotableTask interface which is guaranteed to be offloaded to an Android server. If a class contains only Java code which does not access any Android API, then the class should implement RemotableGenTask interface, Jade can offload such class to more servers (i.e., any server with Java VM). With the exception of different target servers, RemotableGenTask and RemotableTask have similar mechanisms that support code offloading. Therefore, we used RemotableTask interface to demonstrate how it works.

In Jade, a class that implements the RemotableTask interface is called a remotable class. An instance (object) of remotable class is called a remotable object. A remotable object can be executed on the client (locally) or on the server (remotely). An application developed using the Jade programming model is called a Jade compatible application. At runtime, a Jade compatible application contains remotable objects that can run concurrently on multiple servers.

Some types of code must be executed locally. If a class contains the following code, it should not be considered for offloading:

- Code that creates user interface of the application.
- Code that handles user interaction (e.g., callback method for clicking a button).

- Code that accesses the client's special hardware which may be unavailable on the server (e.g., some smart phones are equipped with GPS sensor, but most computers are not).
- Code unsuitable for re-execution (e.g., code that performs online transactions).

Our goal in providing the RemotableTask interface is to eliminate the need for developers to guess whether or not a class is suitable for offloading in terms of energy consumption and performance. When a class does not contain the above code, it can implement the RemotableTask interface, and the Jade runtime engine automatically determines if the class is suitable for offloading.

The RemotableTask interface is the key construct in the Jade programming model, it defines the following methods called by the Jade runtime engine in sequence:

- preExecution, performs preparations before executing the main task (e.g., connect database, initialize data).
- loadData, loads data from the client's file system. If loadData is called on the server, data is read and transferred from the client.
- execution, performs the main task.
- updateData, the counterpart of loadData. After the task finishes, updateData updates data in the client's file system. If updataData is called on the server, data is transferred back to the client.
- postExecution, the counterpart of preExecution that performs remaining tasks after the main task is complete (e.g., disconnect database, update log and send notification to user).

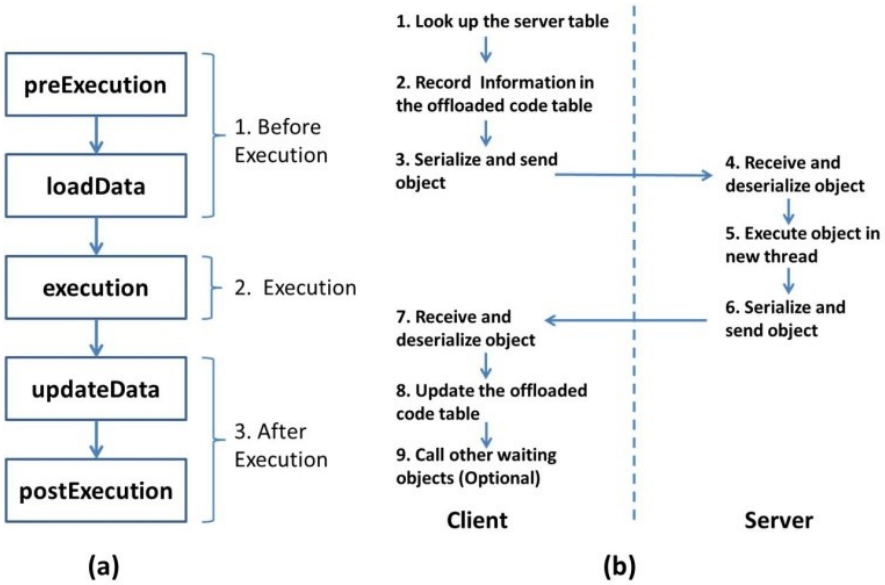
By implementing the RemotableTask interface, the life of a remotable object is divided into three stages (Figure 3-4): before execution, execution, and after execution. This division naturally matches steps of a common computation: 1) preparation (e.g., load data, connect to network); 2) execution, which performs computation on the data; and 3) update, which finishes the execution (e.g., update database, notify user).

The Jade optimizer considers all remotable objects of an application for offloading. If a remotable object is eligible for offloading, the Jade runtime engine handles the code offloading by following the workflow shown in Figure 3-4.

Application development using Jade programming model is intuitive and simple, developers only need to follow two steps:

1. Identify tasks in the application that can be offloaded (code does not violate previously mentioned rules).
2. For each task, create a class that implements the RemotableTask interface.

Figure 3-4 (a) Life of a remotable object is divided into three stages by im-plementing the RemotableTask interface (b) Code offloading workflow



For example, an application can potentially perform three tasks: 1) collect information entered by the user (e.g., name, phone number, address and company); 2) verify the information; and 3) save the information to the database on a remote server. Task three is a good candidate for computation offloading. Using the Jade programming model, we create a remotable class (UpdateInfo) for Task three. The pseudocode is:

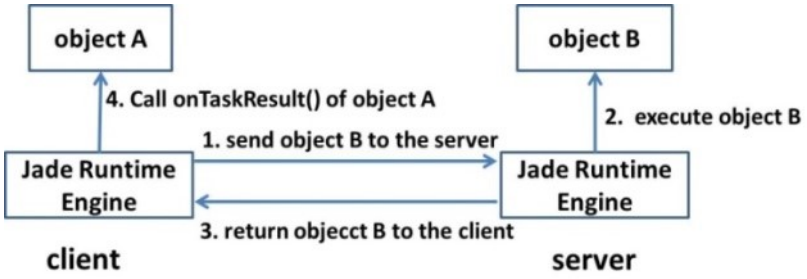
```
class UpdateInfo implements RemotableTask{
    preExecution(){
        open database connection;
    }
    loadData(){
        do nothing;
    }
    execution(){
        if new user
            insert user info into database;
        else
            update user info;
    }
    updateData(){
        do nothing;
    }
    postExecution(){
        close database connection;
        send notification to user;
    }
}
```

In Jade, a remotable object should perform tasks inde-pendently with no dependencies between remotable objects. Advantages of this requirement include 1) reduction of cost model complexity so the optimizer consumes less energy and 2) scalable remote task offloading in

which the number of offloaded tasks can dynamically scale up/down with adding/removing of servers.

The Jade programming model also provides a mechanism for a remotable object to notify local objects when the remotable object finishes execution (Figure 3-5). If a class needs to be notified, the class can implement the OnTaskReturnedListener interface. This interface defines the onTaskResult callback method called by the Jade runtime engine when a specified remotable object finishes execution.

Figure 3-5 Jade programming model provides a mechanism for object B to notify object A when object B finishes execution. Object A must implement the OnTaskReturnedListener in order to receive the notification.



3.5 Evaluation

In this section we evaluate Jade's ability to reduce energy consumption for mobile applications. We implemented two applications that perform common tasks widely used by users.

Image processing, such as image editing and object recognition, is used in many applications. Our first application was FaceDetection (FD) which asks users to select pictures from the device's photo gallery. FD then detects human faces in those pictures and highlights them by putting a rectangle around each face. Code performing face detection was implemented in a remotable class in order to be offloaded to the server.

Navigation application is one of the most popular mobile applications. Path finding in such applications could be computation intensive, making it suitable for computation offloading. Our second application was FindRoute (FR). This application simulates a real world navigation application by determining the shortest path between two nodes of a graph. We used Dijkstra's algorithm for path finding, and path finding code was implemented in a remotable class.

We used a Moto X smart phone as the client. Moto X is equipped with Qualcomm Snapdragon 1.7GHz Dual-Core CPU, quad-core GPU, and 2GB RAM. We used a Samsung Galaxy S3 smartphone and a Dell Inspiron 15 laptop as the servers. Galaxy S3 has a Quad core 1.4GHz processor and 1GB RAM. Inspiron 15 has a 1.8GHz Dual-Core CPU and 8GB RAM. All devices were connected in the same wireless network.

In order to evaluate how much energy Jade can save for mobile device, we run FD and FR on Moto X. FD performs face detection on 50 pictures, the size of each picture is under 200KB. Each application was executed twice: first time with Jade disabled and second time with Jade enabled. We compared the first execution with the second execution to evaluate the performance of Jade.

Results showed that Jade effectively reduces power consumption for FD (Figure 3-6). Average power consumption was reduced by 34% (Figure 3-10). FR had similar results (Figure 3-8): Jade reduced 39% of average power consumption for FR. Power reduction was achieved because when Jade was enabled, some computations were offloaded to servers, thereby, reducing the workload of the client (Figure 3-7 and 3-9). Because more tasks were executed concurrently, Jade also improved the performance of both applications: it reduced 37% and 45% of execution time for FD and FR respectively (Figure 3-11).

Results demonstrated that Jade can effectively reduce energy consumption of mobile applications and improve application performance.

Figure 3-6 Power consumption of FaceDetection

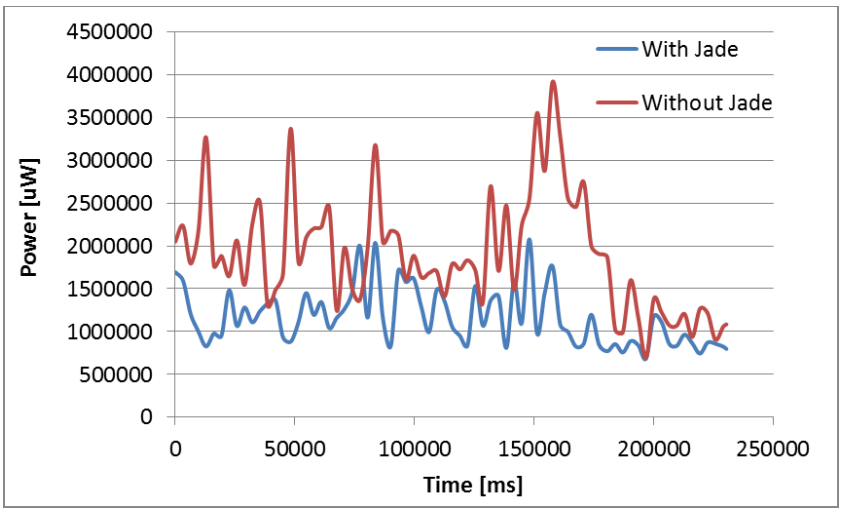


Figure 3-7 CPU load of FaceDetection

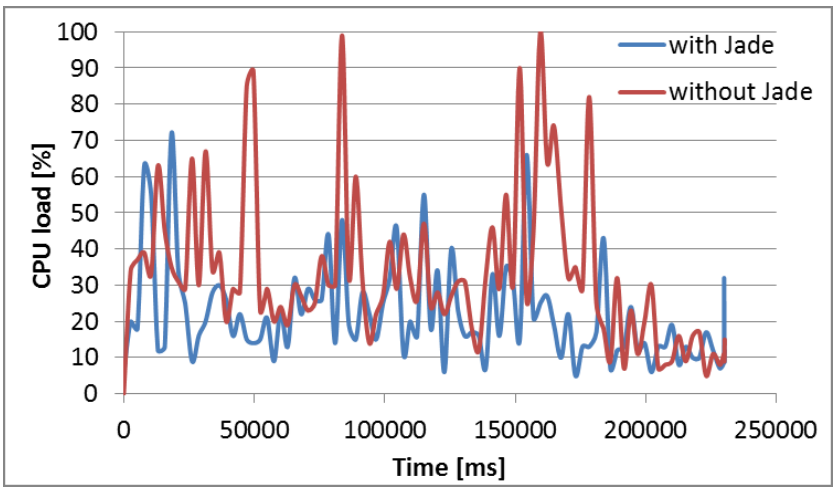


Figure 3-8 Power consumption of FindRoute

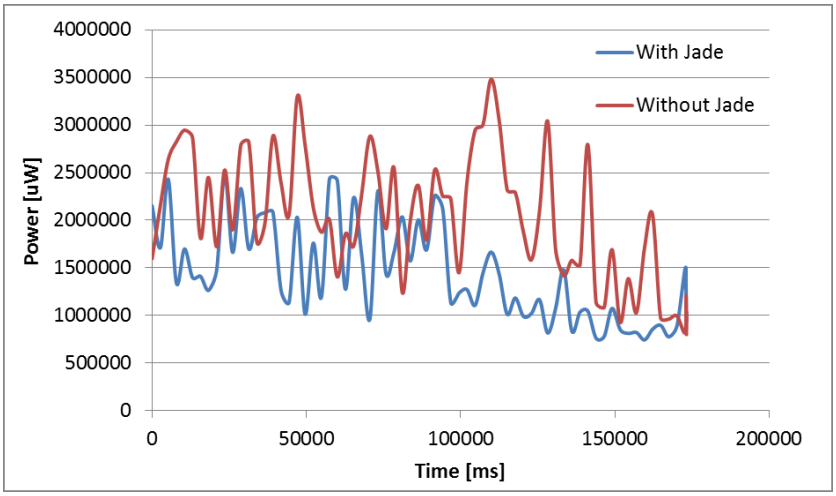


Figure 3-9 CPU load of FindRoute

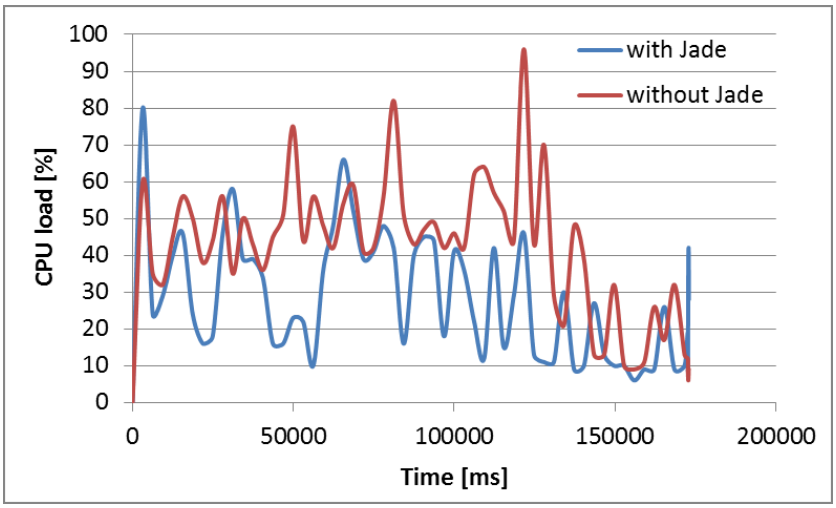


Figure 3-10 Average power consumption of FaceDetection and FindRoute

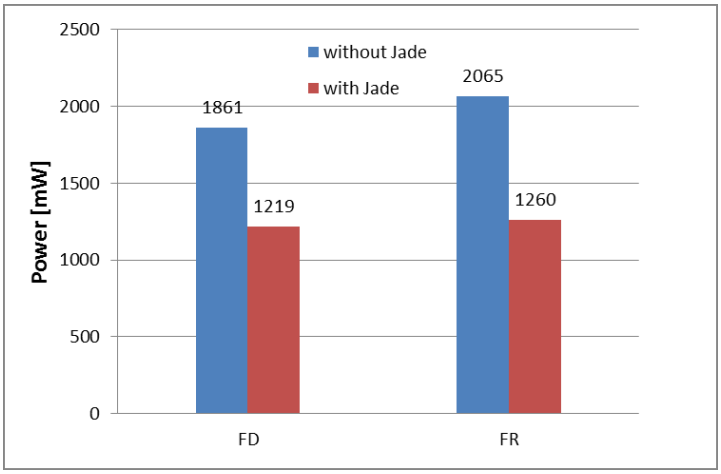
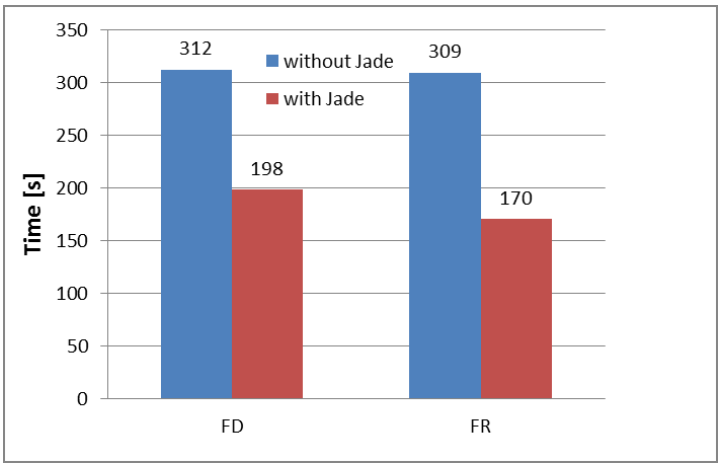


Figure 3-11 Execution time of FaceDetection and FindRoute



3.6 Related Work

Mobile devices have limited resources such as battery capacity, storage, and processor performance. Computation offloading is an effective method to alleviate these restrictions by sending heavy computations to resourceful servers and receiving results from these servers. Many issues related to computation offloading have been investigated in the past decade, including feasibility of offloading, offloading decisions, and development of offloading infrastructures.

Finding energy efficient task scheduling algorithms has been one of hot research topics in the area of high performance systems because of the huge energy cost. Since there are many jobs running in such systems, efficient power management is a critical problem. Recently, research in high performance computing has also introduced and developed power-aware algorithms to reduce the total energy not only for the operational cost but also for the system Reliability [68 - 81].

Jade was built upon previous research regarding program partitioning, code offloading, and remote execution. In this section, we provide an overview of proposals by these researches and how they relate to Jade.

Cuervo et al. proposed MAUI [39], a system that enables energy-aware offloading of mobile code to the infrastructure. MAUI enables developers to produce an initial partitioning of their applications by annotating methods and/or classes as remotable. At runtime, the MAUI solver decides which remotable methods should execute locally and which should execute remotely. Unlike MAUI, Jade provides a sophisticated programming model with a full set of APIs, so developers have total control on: how application is partitioned, where code is offloaded and how remotable code interacts with local code. In Jade, dependencies do not exist between remotable tasks, the profiler and optimizer do not need to analyze the whole program, thereby, energy cost of program profiling and cost model calculation is lower than MAUI.

The energy efficiency of mobile clients and the relation with several metrics for characterizing the energy consumption of communication was discussed in [63]. The impacting factor for energy usage include: the amount of transferred data, the energy characteristics of the device's wireless transfer, communication bit-rate, traffic pattern, and whether the client is near or far from the server. However, the author just discussed the various metrics that characterize computation offloading and did not experiment their work in a real offloading system.

Chun et al. proposed CloneCloud [40], an application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. In CloneCloud, threads must be paused, all states of the threads must be transferred to the server, and then threads resume on the server in order to offload computation. Offloading is expensive, however, especially when the client and server are both resource constraint mobile devices. In contrast, code offloading in Jade is lightweight. Remotable objects

are serialized, transferred, and deserialized, resulting in much lower overhead compared to thread migration.

Some software-based solutions were also investigated besides computation offloading, such as applying power saving modes on processes, network interface cards, display, and sensors [64]. There were also hardware-based solutions such as providing low power processor for low energy usage tasks and high power processor for energy intensive tasks [65]. Furthermore, some research proposed charging the phone from different surrounding sources such as light, speech, movement, bacteria, and human heart's beats [66, 67].

3.7 Future Work

In our future work, we will extend Jade to include cloud platform. Cloud provides a scalable and powerful computing environment which is ideal for complex computing tasks. Today, many mobile devices have fast wireless link such as 4G LTE without limitation of locations, making cloud platform a good destination for computation offloading. We will also study how cloud changes the development pattern of mobile applications.

Another direction of future work is security [100 - 104]. Jade offloads computation to trusted devices, but security concerns arise if computation offloading is scaled up, for example, multiple applications accessing a single server, running foreign code on the server, and remote codes interfering with each other.

3.8 Conclusion

In this paper, we presented Jade, a system that enables computation offloading for mobile devices. Jade can effectively reduce energy consumption of mobile devices, and dynamically change its offloading strategy according to device status.

We evaluated Jade with two applications: a face detection application and a path finding application. Results showed that Jade can effectively reduce energy consumption for both applications while improving their performance.

Chapter 4 - An Energy-saving Task Scheduler for Mobile Devices

This chapter is published in [123] and reprinted in full in accordance with the IEEE rules.

4.1 Introduction

Mobile devices, such as smart phones and tablets, have become a necessity because they allow people to perform a wide variety of useful activities (e.g., video calls, emails, gaming, social networking, navigation, etc.) with mobility. These devices typically are equipped with a relatively powerful mobile processor, a rich set of sensors, and a substantial amount of memory. It allows previously unimaginable applications to be developed by integrating many sensors, such as motion sensors, position sensors, and environmental sensors.

However, battery life has become one of the biggest ob-stacles for mobile device advancements. Performance de-manded by smartphones and tablets is increasing at a much faster rate than technological improvements in battery capacity. The need for increased performance of mobile devices directly conflicts with the desire for longer battery life.

One popular technique to reduce energy consumption of mobile devices is computation offloading in which an application reduces energy consumption by delegating code execution to other devices. Traditionally, computations are offloaded to remote, resource-rich servers. Selection of a proper offloading strategy can reduce power consumption and simultaneously enhance performance for mobile device.

Nowadays, people can easily access multiple computing devices, e.g., desktop and laptop computers at office and home, even when people are traveling, they bring smartphones and tablets. So for a computation offloading system, it’s not hard to find suitable servers. With the help of faster wireless technology like 4G, cloud platform also becomes a good destination for computation offloading. Cloud platform provides a scalable and powerful computing

environment which is ideal for complex computing tasks. Cloud enables computation offloading without the limitation of location.

In this paper, we present Jade [45 -51], an energy-aware computation offloading system for mobile devices. Jade, built for mobile devices running Android operating system, minimizes energy consumption of mobile devices through fine-grained computation offloading to the cloud. Jade provides the following benefits:

1. A runtime engine that enables computation offloading from mobile device to the cloud. By monitoring application and device status, the runtime engine automatically decides if code should run locally or remotely.
2. A programming model that helps developers creates applications with computation offloading ability.
3. Jade supports multiple servers, i.e., computation can be offloaded from mobile device to mobile device, desktop, laptop and cloud (Figure 4-1).

Running on mobile device, Jade monitors the energy consumption of apps and offloads energy consuming computations to servers. It effectively reduces the amount of computation on mobile device in order to extend the battery life of device. Jade is effective under many use cases, here are two examples:

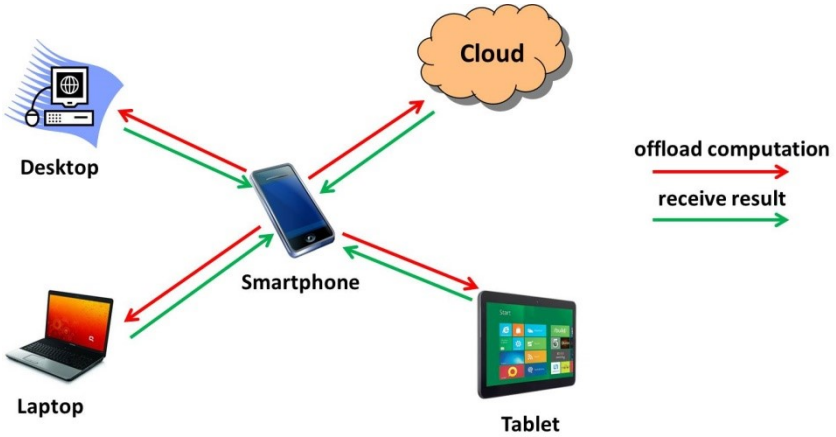
1. Nowadays, people bring multiple mobile devices when they travel, e.g., smartphone and tablet. Longer smartphone battery life is desired during the travel, because people want to stay connected with others, and it's usually not easy to re-charge the phone on the way. With the help of Jade, energy demanding computations are offloaded from smartphone to less used devices (e.g., tablet) in order to reduce the energy consumption. Thereby, users can enjoy using their smartphone without worrying about its battery life.

2. People working in the office spend a lot of time sitting in front of their computer. For users who use their smartphone a lot, they often need to re-charge the phone during the day. By enabling Jade, computation intensive code are offload from the smartphone to more powerful computers, so users don't need to re-charge their phone in the middle of the day.

Our contributions are summarized as follows:

- We present the design and implementation of a complete system. Jade is able to offload computation from an Android device to servers running on the cloud.
- We present a multi-level data storage strategy that optimizes the energy cost of data transfer when computation offloading occurs.
- We present a multi-level scheduling algorithm that automatically and seamlessly transports workloads to the appropriate server based on task's performance and energy needs.
- We evaluate Jade with two applications. Results indicated that Jade can effectively reduce up to 40% of average power consumption for mobile device, while reducing the execution time of applications.

Figure 4-1 Jade can offload computation from mobile device to multiple servers



4.2 System Design

In this section, we present the high-level design of Jade and its programming model in order to demonstrate how they integrate into one system, thereby supporting distributed execution of mobile applications.

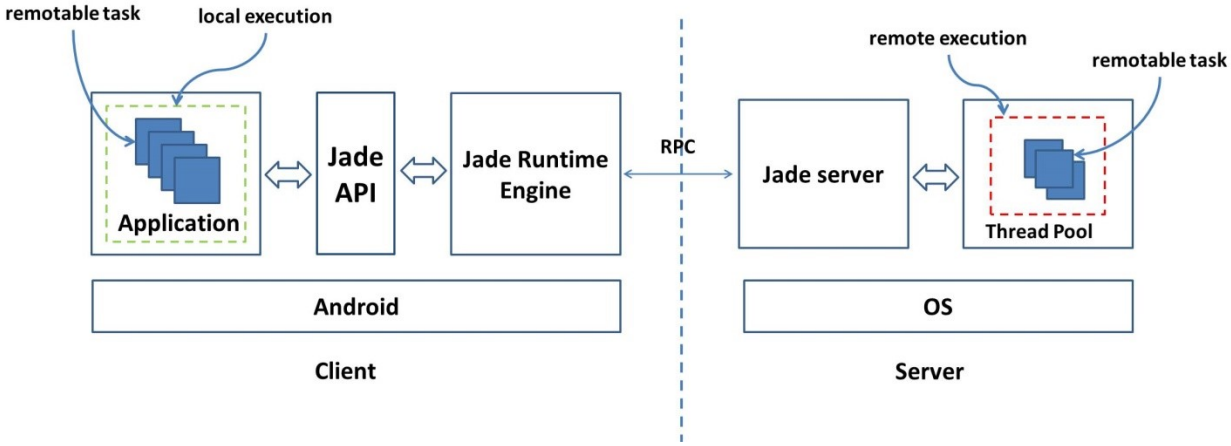
For mobile applications with heavy computation needs, computation offloading is an effective method to reduce energy consumption and enhance performance. However, it's not easy to build computation offloading ability into mobile apps, the reasons includes:

- It requires additional efforts and skills to develop applications with computation offloading ability and, unfortunately, no mature frameworks or tools exist for mobile application developers.
- Most mobile developers (especially independent developers) don't have expertise on profiling and optimizing energy usages of mobile app, and they only focus on the functionality of the app without paying attention to its energy usage, e.g., users often report that some apps have abnormal energy usage and kill the battery very fast.

We designed Jade to simplify the process of building energy efficient apps, and also minimize the workload of developing such apps by:

- Offering the Jade runtime engine which provides services for wireless communication, device profiling, program profiling and computation offloading. Conceptually, the Jade runtime engine automatically transforms application execution on one mobile device into a distributed execution optimized for wireless connection, power usage, and server capabilities.
- Providing an easy-to-use programming model for developers to build mobile applications that support energy-aware computation offloading.

Figure 4-2 Jade enables computation offloading for mobile applications. Applications contain remotable tasks that can be offloaded from the client to the cloud. Jade runtime engine automatically decides where to execute remotable tasks and initiates distributed execution.



In order to increase understanding of this offloading system, terms used in Jade must be defined. The mobile device that offloads computation is called the client. The Jade server that receives and executes the offloaded code is called the server. Servers run on the cloud. Mobile applications contain remotable tasks which can be offloaded to the server (Figure 4-2). If a remotable task is executed on the client (i.e., it is not offloaded), we call it local execution. In contrast, if a remotable task is executed on the server (i.e., it is offloaded), we call it remote execution.

4.2.1 Jade Runtime Engine

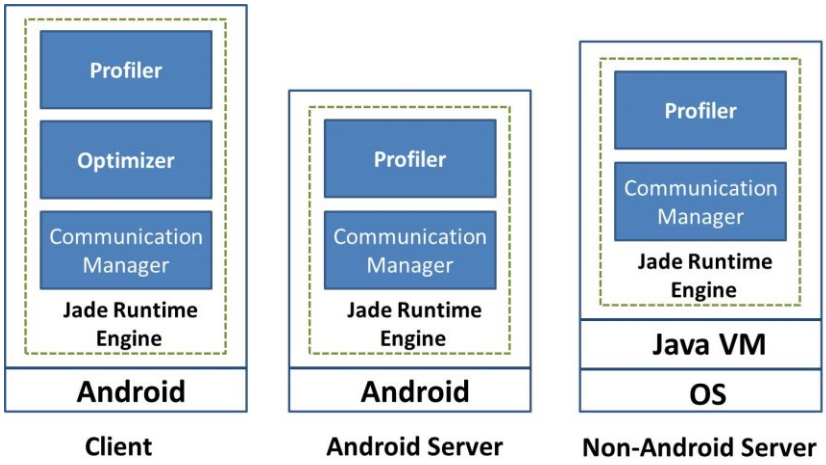
In order to support computation offloading for applications, a computation offloading system is needed to handle some essential tasks such as:

- Communication. In order to offload code from the client to the server, the system should be able to 1) connect to the server; 2) send data between client and server; 3) coordinate with server for distributed execution; 4) track status of remote execution; 5) restore execution if unexpected errors occur (e.g., wireless connection lost, remote execution

failure); and 6) exchange and record information related to client and server (e.g., connection speed, CPU usage, battery level, hardware configuration).

- Profiling. In order to make correct offloading decision, the system should have fresh information regarding status of the device and application. Device profiling is the process of collecting information pertaining to device status, such as wireless connection, CPU usage, and battery level. Similarly, program profiling collects information about applications, such as execution time, energy consumption, memory usage, and data size.
- Optimization. The system should be able to find an optimized offloading strategy in order to maximize application's energy savings and performance.

Figure 4-3 High-level design of the Jade runtime engine



The goal of Jade is to maximize the benefits of energy-aware computation offloading for mobile applications, at the same time, minimizing the burden on developers to build such an application. By proving the Jade runtime engine, computation offloading is handled automatically in the background, allowing application developers to focus on the application without implementing a computation offloading system.

Jade runtime engine components are shown in Figure 3. On the client device, the Jade runtime engine runs as a group of background services. On the cloud, Jade server also contains a

Jade runtime engine. Jade server is platform independent, it can run in any virtual machines on the cloud.

Section 5 details the components of the Jade runtime engine.

4.2.2 Jade Programming Model

In addition to the Jade runtime engine, another key contribution of our system is the Jade programming model which helps developers efficiently build applications with computation offloading ability.

The key component in the Jade programming model is the Jade API. We assume that most mobile developers don't have knowledge on energy profiling and they don't spend much time optimizing their code to be energy efficient. So the idea behind Jade API is that developers don't need to guess if a piece of code is energy demanding and suitable for offloading, all they need to do is annotating that code. When the code is executed, Jade will decide if it should be offloaded or not. Compared with some previous work, Jade greatly reduces the requirement and workload for developers to build energy efficient apps.

Other attribute of Jade API is that it follows the development logic of Android and Java, it introduce no strange rules and concepts. For developers new to Jade, they don't need to spend much time learning, the API is very straightforward and easy to use. Developers become comfortable using it immediately.

Details of the Jade programming model are provided in Section 5.

4.3 Muti-level Data Storage

When computation offloading occurs, the dominant energy cost is the energy consumption of the network interface. Wireless network interface accounts for a big portion of mobile device's energy consumption. For example, the Wi-Fi interface of a smart phone

consumes almost 1500mW when downloading data [42]. Because computation offloading system needs to transfer data between client and server frequently, a lot of energy of mobile device is consumed by network interface. Thereby, reducing the amount of data transferred between client and server is crucial in order to save energy for mobile device.

Jade provides a Multi-level Data Storage Service (MDSS) that optimizes the energy cost of data transfer when computation offloading occurs. MDSS allows developers to save application data on the cloud without writing any backend code. Application data is also saved locally on mobile devices allowing applications to work when devices are offline. MDSS automatically synchronizes data between local device and the cloud, so developers can focus on creating applications instead of having to worry about building backend solution to handle data storage and synchronization.

When application generates new data, MDSS first saves the data on local device, so the data is always accessible to application. Data is uploaded to the cloud later when MDSS performs synchronization. Synchronization of data sets between client device and the cloud can be triggered by calling the synchronize method or charging the device. In order to synchronize, MDSS reads the latest version of the data available in the cloud and compares it to the local copy. After comparison, MDSS writes the latest updates as necessary to the local copy and the cloud. MDSS maintains the last-written version of the data by default.

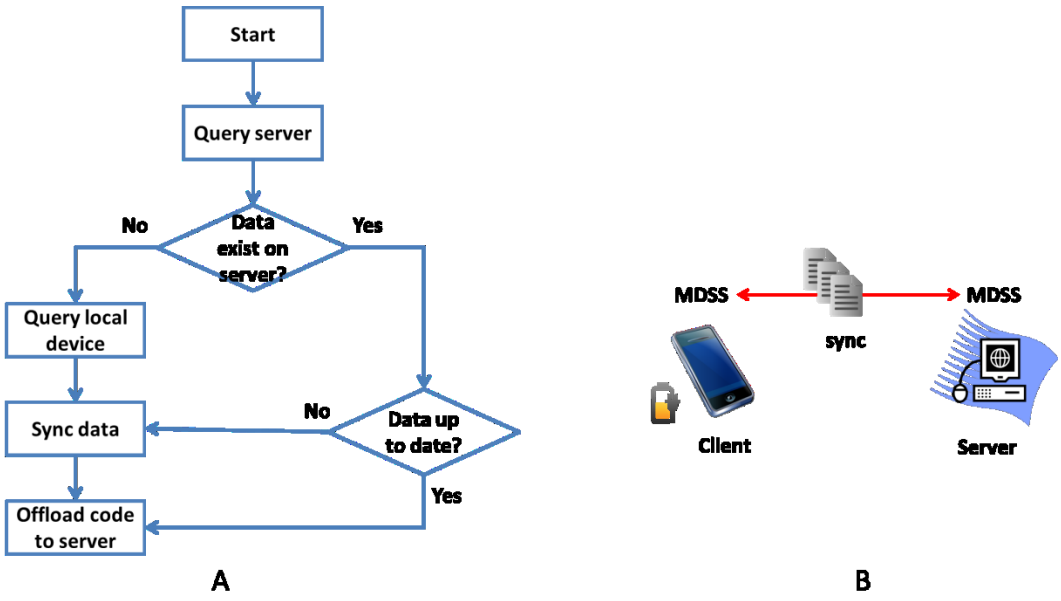
In a computation offloading system, a remotable task contains two elements: 1) app data (e.g., images, texts, numbers) and 2) task code that performs execution on app data (e.g., sorting algorithm, image processing). App data and task code are bundled and transferred when a remotable task is offloaded to the server. In most cases, the size of app data is much bigger than the size of task code (e.g., size of an image could be a few MB, whereas size of task code

performing complex computation could be a few KB). By introducing MDSS, a remotable task's app data and task code are separated. In Jade, a remotable task i contains only task code, the app data accessed by i is stored separately and referenced by URI. When i is offloaded to the cloud, if the cloud already has the most recent copy of app data that i needs to access, Jade only offloads task code to the cloud in order to reduce the amount of data transferred. MDSS effectively helps reducing the energy consumption for mobile device's network interface by avoiding transferring app data every time when i is offloaded.

Jade uses URI to reference the app data to be acted on. When a remotable task i is chosen for offloading, URI of i's app data is passed to MDSS which then queries the data using the URI. If the latest version of the app data is found on the cloud, Jade offloads i to the cloud without synchronization. If the cloud does not have the app data or the cloud has an older version of the app data, MDSS synchronizes the cloud with local device before offloading i (Figure 4-4A).

Jade provides APIs to let developers control when to sync data according to varying needs. By default, MDSS automatically performs synchronization only when device is charging in order to preserve battery life (Figure 4-4B). Synchronization can be triggered anytime in code by calling the synchronize method. Developers can also control synchronization according to device's wireless connection. For example, in order to save user's mobile data plan, synchronization can be disabled when device does not have Wi-Fi access.

Figure 4-4 (A) Data sync workflow in MDSS (B) MDSS automatically synchronizes data between client and server when client is charging



MDSS enables interesting new design for mobile apps which can't be achieved easily before. By adopting MDSS, many existing apps can be modified so they become more energy friendly. Here is one example: in Motorola's flagship device Moto X, there is an interesting feature called Highlight Reel. Every day, it randomly chooses some pictures taken by user during that day and creates a small video. It's a great idea but video rendering is energy consuming. With MDSS, the album of Moto X can be synchronized with cloud when device is charging. Since the pictures are already available on the cloud, Jade only offloads the video rendering code to the cloud where the video is created and sent back to Moto X. This way, users can also have a great video generated everyday but with less energy consumption on their device.

4.4 Implementation

This section highlights important implementation details of Jade. Sections 5.1 and 5.2 describe components in the Jade runtime engine. Section 5.3 shows details of the Jade programming model.

4.4.1 Profiler

When a remotable task is generated at run time, the Jade optimizer determines whether it should be executed locally or remotely. This decision is based on information provided by the profiler. The profiler collects the following information: 1) device status, such as charging status, battery level, CPU load, and wireless connection status); and 2) characteristics of remotable tasks, such as execution time, size, and energy consumption. The profiler measures characteristics of a remotable task during task's first execution and continuously monitors the status of devices because mobile device's status changes frequently (e.g., loss of Wi-Fi connection, reaching low battery level). The optimizer may make wrong decision on whether code should be offloaded based on a stale measurement.

Offline and online methods are available to measure power consumption for mobile devices and applications [82-90]. An offline method, often used under laboratory conditions, uses external measurement tools [91-99]. This method generates accurate results but requires special skills and equipment, most developers and users cannot realistically perform offline power measurement. The online method overcomes limitations of the offline approach by utilizing the Battery Monitoring Unit (BMU), which provides information related to battery (e.g., supply voltage, current, and battery temperature). A majority of current mobile devices are equipped with BMU. However, online method may not be as accurate as offline method because the information update rate of BMU is much lower than external measurement tools. According to previous research of mobile device power modeling, online method can generate satisfying results. The Jade profiler uses the online approach to measure energy consumption for devices and applications. The evaluation showed that Jade can make correct offloading decision when using information generated by this approach.

In Jade, we use Trepn Profiler (Trepn) [41] to measure the energy consumption of the device and apps. Trepn is an on-target power and performance profiling application for mobile devices. It runs on most Android devices featuring Qualcomm Snapdragon processors or development hardware. Trepn has the following features:

- Overlays appear on screen on top of applications that are being profiled
- Profile device, or a single app
- Displays battery power on supported devices
- View CPU and GPU frequency and utilization
- Show GPU frequency and utilization
- Display network usage (cellular and Wi-Fi)
- Runs on most Android smartphones and tablets (Android 4.0 and higher)

With Trepn, developers can better understand the impact of their programming choices on both power and performance.

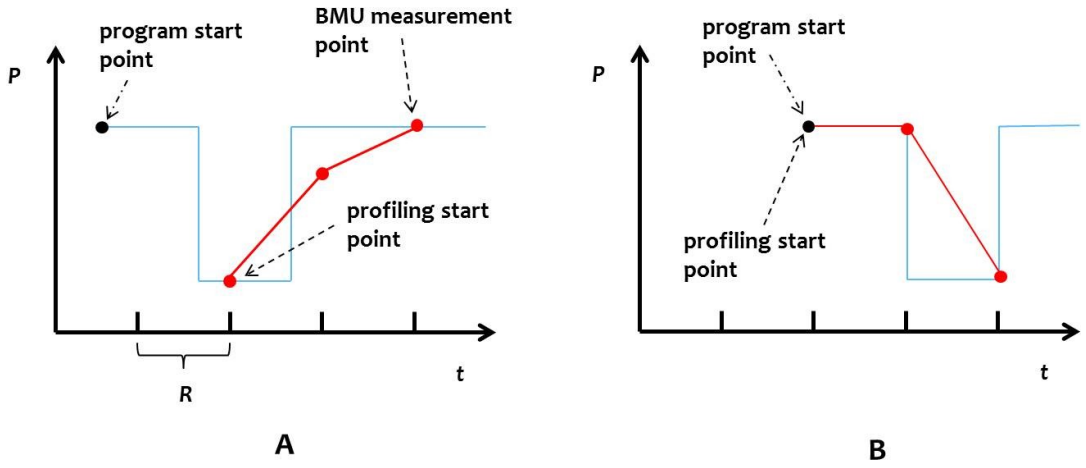
4.4.1.1 Synchronized App Profiling

When a new remotable task is generated, a unique name is assigned to it (we combine the package name of the application and class name of the remotable task to form a unique name). During the remotable task's first execution, the profiler collects the task's energy consumption E, execution time T, and size S. Then the information is provided to the optimizer. Based on the cost model, the optimizer decides if the task is suitable for offloading. Finally, all information concerning the remotable task is recorded in a database. When the same or similar task is executed again next time, Jade finds its information in the database in order to understand whether or not the task should be offloaded.

With our automatic program profiling, developers only need to identify potential remotable tasks in a program and annotate these tasks as remotable. When the application is installed on a mobile device, developers only need to run the application, invoking as many remotable tasks as possible. Jade automatically analyzes remotable tasks in the background and decides whether or not they are eligible for remote execution.

We improved the profiling accuracy of the Jade profiler compared with previous versions. In order to obtain a more accurate power measurement of a remotable task, we need to synchronize the profiler and the remotable task. In Figure 4-5, the blue line represents the actual power of the task, the red line represents the reading of BMU. In Figure 4-5A, the profiler starts later than the program, so the reading of BMU is less than the actual value. The BMU reports more accurate value if the profiling and program starts at the same time (Figure 4-5B).

Figure 4-5 The start time of remotable task and profiler need to be synchronized in order to obtain accurate power measurement



In order to synchronize profiler and the program, we implemented a controller in the profiler which performs several jobs:

- Collect the information about the BMU (e.g., update rate R)
- Suspend/release the execution of a remotable task to be measured

- Start/terminate the execution of profiler

By manipulating the start time of program and profiler, the controller guarantees that the profiler starts at the same time with the program to be measured, thereby generating more accurate power measurements.

4.4.1.2 Device Profiling

At runtime, the profiler also continuously monitors device status. Collected information includes battery (i.e., battery level, charging status), wireless connection (i.e., Wi-Fi turned on/off, throughput), and CPU load.

Due to the nature of mobile devices, wireless connection could change frequently because users often change locations. Fresh information about wireless connection is crucial in order for the optimizer to make correct offloading decisions. Similar to MAUI, we used a simple method to measure the wireless link: Each time Jade offloads code, the profiler measures the transfer duration in order to obtain more recent average throughput. This simple approach takes into account both the latency and bandwidth characteristics of the network. We also built a simple energy cost model for wireless transfer using this approach: we send synthetic data from the client to the server, varying the size of the data, and we measure energy consumption of each transfer. This model allows us to predict energy consumption of data transfer as a function of the size of data.

4.4.2 Optimizer

The purpose of the Jade optimizer is to choose suitable remotable tasks to offload to the server in order to find an offloading strategy that minimizes application's energy consumption.

The optimizer makes the offloading decision by solving an optimization problem using information provided by the profiler.

Characteristics of a remotable task determine where it should be executed. The code should be offloaded only if the energy cost to execute it locally is greater than the energy cost to transfer it. For example, code performing heavy computation on small data is suitable for offloading. For some code, the energy cost of transfer outpaces the energy cost of local execution (e.g., lightweight computation on big data), such code should not be offloaded.

Based on information provided by the profiler, the optimizer determines the best offloading strategy by solving the following cost model. I represents the set of remotable tasks in the application. For each remotable task $i \in I$, E_i^e is energy consumed to execute i locally. E_i^t is energy consumed to transfer i between the client and the server. T_i^l is the execution time of i on the client. T_i^r is the execution time of i on the server. T_i^t is the transfer time of i. I_i is the indicator variable: $I_i = 0$ if i is executed locally, $I_i = 1$ if i is executed remotely. The optimizer must find the assignment for I_i such that:

- *maximizes* $\sum_{i \in I} I_i \times (E_i^e - E_i^t)$
- *guarantees* $\sum_{i \in I} (1 - I_i) \times T_i^l + I_i \times T_i^r + I_i \times T_i^t \leq l$

The first formula is total energy savings. The second constraint stipulates that the total execution time is within l. Developers can configure l according to their specific requirements.

As explained in Section 5.4, the Jade programming model requires that every remotable task i to perform independent job, so no dependencies exist between remotable tasks. This requirement further simplifies the cost model. For each remotable task i, the optimizer must solve only the following inequation:

$$E_i^e - E_i^t > 0$$
$$T_i^r + T_i^t - T_i^l \leq l$$

$E_i^e - E_i^t$ is the energy saving if i is executed remotely. i should be considered for offloading only when $E_i^e > E_i^t$. Similarly, the second inequation guarantees that time difference between remote execution and local execution is not greater than l.

4.4.3 Jade Programming Model

When designing applications that support code offloading, the application must be partitioned into sub-parts which can be offloaded to the server. Applications can be partitioned at various levels, such as class, method, process, and thread. An application is partitioned at the class level in Jade. Developers can produce an initial partition of their applications with minimal effort using the Jade programming model.

To be considered for offloading, a class must implement the RemotableTask interface. In Jade, a class that implements the RemotableTask interface is called a remotable class. An instance (object) of remotable class is called a remotable object. A remotable object can be executed on the client (locally) or on the server (remotely). An application developed using the Jade programming model is called a Jade compatible application. At runtime, a Jade compatible application contains remotable objects that can run locally or remotely.

To partition a program, developers need to follow some rules. Three properties of any legal partition are explained here:

Property 1. Code that access special hardware of the local device can't be offloaded.

If remotable task uses special resources such as GPU or other hardware accelerator, the code must remain locally. This property guarantees that the partitioned program has a better compatibility with other devices.

Property 2. Code that 1) creates user interface of the application; 2) handles user interaction (e.g., callback method for clicking a button) can't be offloaded.

Code related to UI rendering needs to be responsive in order to provide a smooth user experience, this property guarantees that offloading doesn't impact the UI rendering.

Property 3. Code unsuitable for re-execution (e.g., code that performs online transactions).

Property 4. Nested offloading is not allowed.

This implies that nested suspends and nested resumes are not allowed. Once the code is suspended for offloading at some point, the code should not suspend again before resume, i.e., migration and re-integration should happen alternately.

Our goal in providing the RemotableTask interface is to eliminate the need for developers to guess whether or not a class is suitable for offloading in terms of energy consumption and performance. When a class does not contain the above code, it can implement the RemotableTask interface, and the Jade runtime engine automatically determines if the class is suitable for offloading.

The RemotableTask interface is the key construct in the Jade programming model, it defines the following methods called by the Jade runtime engine in sequence:

- sychronize, invoke MDSS to check if app data is available on the server, if not, synchronize the data.
- execution, performs the main task.
- postExecution, anything follows the main task should be done here, e.g., release some resources, synchronize data between devices.

By implementing the RemotableTask interface, the life of a remotable object is divided into three stages: before execution, execution, and after execution. This division naturally matches steps of a common computation: 1) preparation (e.g., load data, connect to network); 2)

execution, which performs main task; and 3) update, which finishes the execution (e.g., update database, notify user).

The Jade optimizer considers all remotable objects of an application for offloading. If a remotable object is eligible for offloading, the Jade runtime engine handles the code offloading by following the workflow shown in Figure 4-6.

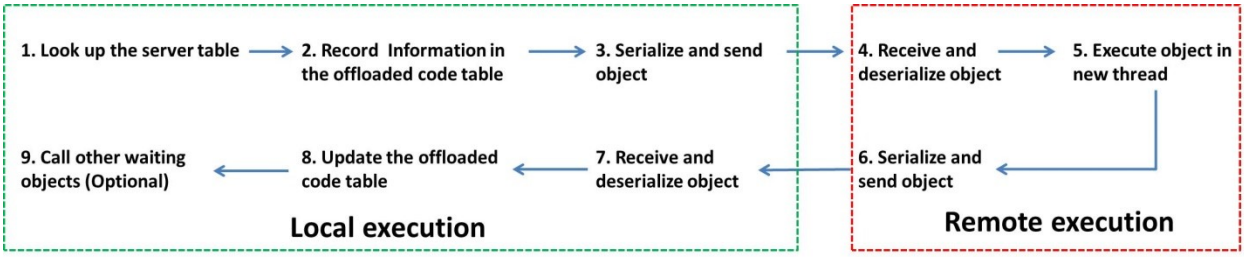
Application development using Jade programming model is intuitive and simple, developers only need to follow two steps:

1. Identify tasks in the application that can be offloaded (code does not violate previously mentioned rules).
2. For each task, create a class that implements the RemotableTask interface.

For example, an image editing app can detect the human faces appearing in a photo. The face detection code is suitable for offloading. Using the Jade programming model, we create a remotable class (FaceDetection). The pseudocode is:

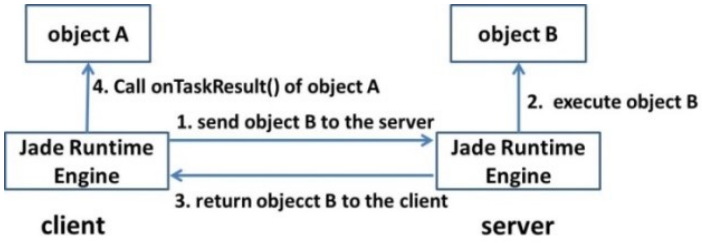
```
class FaceDetection implements RemotableTask{
    synchronize(){
      // invoked MDSS to make sure data is available on the server
    }
    execution(){
      // the face detection code goes here
    }
    postExecution(){
     // perform synchronization
     // release any resources if needed
    }
}
```

Figure 4-6 Code offloading workflow



The Jade programming model also provides a mechanism for a remotable object to notify local objects when the remotable object finishes execution (Figure 4-7). If a class needs to be notified, the class can implement the OnTaskReturnedListener interface. This interface defines the onTaskResult callback method called by the Jade runtime engine when a specified remotable object finishes execution.

Figure 4-7 Jade programming model provides a mechanism for object B to notify object A when object B finishes execution. Object A must implement the OnTaskReturnedListener in order to receive the notification.



4.5 Evaluation

In this section we evaluate Jade's ability to reduce energy consumption for mobile devices. We implemented two applications that perform common tasks widely used by users.

Image processing, such as image editing and object recognition, is used by many applications. Our first application was FaceDetection (FD) which asks users to select pictures from the device's photo gallery. FD then detects human faces in those pictures and highlights them by putting a rectangle around each face. Code performing face detection was implemented in a remotable class in order to be offloaded to the cloud.

The second application is TextSearch (TS), which searches text files against a library of 1000 key words. It counts how many times each key word appears in the text files. The search algorithm is implemented in a remotable class. At runtime, the text files can be searched locally or remotely.

We used a Moto X smart phone as the client. Moto X is equipped with Qualcomm Snapdragon 1.7GHz Dual-Core CPU, quad-core GPU, and 2GB RAM. Jade is used under two different type of scenarios: 1) code is offloaded to remote server (e.g., cloud) and 2) code is offloaded to servers nearby (e.g., tablet, desktop and laptop). We choose different servers to represent there scenarios. For remote server, we created two optimized virtual machines on the Microsoft Azure cloud platform (Azure) with each virtual machine ran a Jade server. Each virtual machine has two cores and 7 GB ram. The client connected to Azure with 4G. For nearby server, we use a Samsung Galaxy Tab S 10.5" tablet (Tab). The Tab has Exynos 5 Octa (1.9Ghz Quadcore + 1.3 Ghz Quadcore) processor, 3 GB ram and 7900 mAh battery. The Tab was fully charged for the experiments. The client connected to server with WiFi.

We used Trepn profiler to measure the power consumption and performance of Moto X. Trepn profiler is a product of Qualcomm, it is a diagnostic tool that lets developers profile the performance and power consumption of Android applications running on devices featuring Qualcomm Snapdragon processors.

To evaluate how much energy Jade saves for applications, we FD and TS on the client with Jade enabled and disabled, and varying the size of app data. Image file is less than 200KB in FD and text file is 50KB for TS.

Results showed that Jade effectively reduced power consumption for FD, average energy consumption was reduced by 30% (Figure 4-8). TS had similar results (Figure 4-10): Jade

reduced 40% of average power consumption. Power reduction was achieved because some computations were offloaded to servers when Jade was enabled, thereby, reducing the workload of the client. The result also indicates that WiFi interface is more energy efficient than 4G interface, which has been shown in the previous research.

Jade also improved the performance of both applications when computation was offloaded to Tab: it reduced 10% and 15% of execution time for FD and TS respectively (Figure 4-9 and 4-11). There are two reasons: 1) The server is more powerful than the client and 2) More remotable tasks were executed in parallel. We also noticed that when computation was offloaded to Azure, the execution time was longer than local execution. The performance of Jade working with Azure was worse than working with Tab, there are several reasons: 1) Network delay is much higher for 4G than WiFi direct, so the data transfer to Azure time is longer; 2) Our implementation of Jade server for Android device is more efficient than Jade server for non-Android device. In the case of offloading computation to Azure, the execution time penalty was less than 4%, which is still acceptable for our applications.

Figure 4-8 Energy consumption of FD

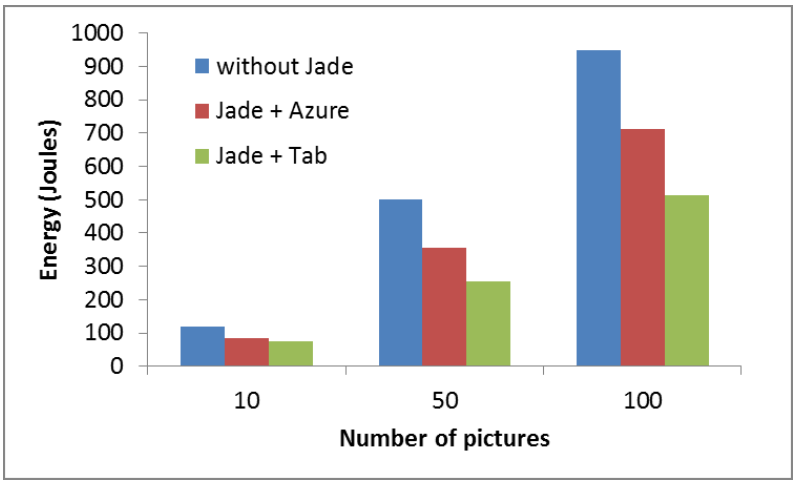


Figure 4-9 Execution time of FD

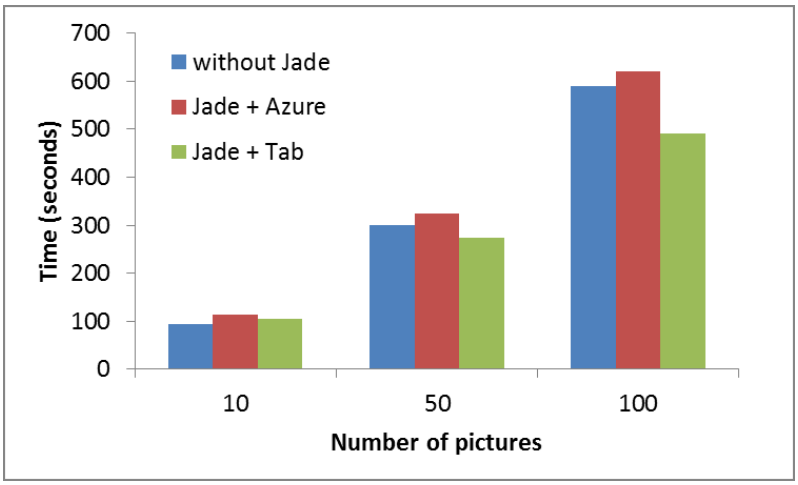
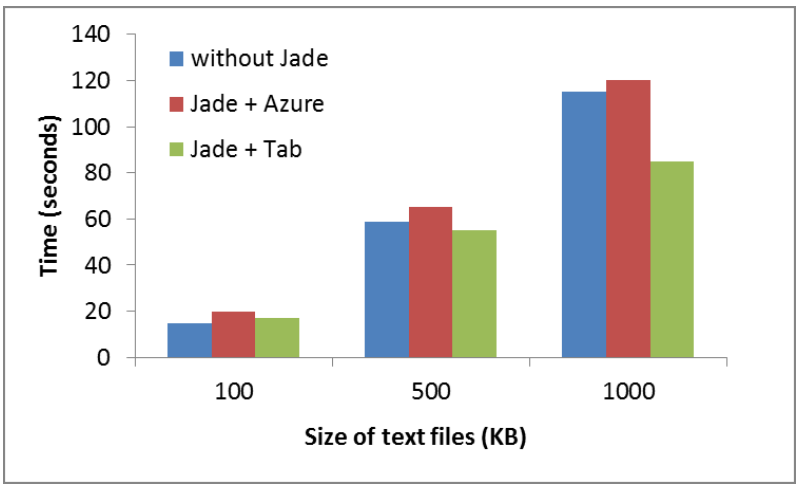


Figure 4-10 Energy consumption of TS

Figure 4-11 Execution time of TS



The results demonstrated that Jade can effectively reduce energy consumption of mobile device and improve application performance.

4.6 Related Work

Jade was built upon previous research regarding program partitioning, code offloading, and remote execution. In this section, we provide an overview of proposals by these researches and how they relate to Jade.

In order to measure whether computation offloading would save energy or not, Kumar et al formulated an equation of several parameters [62]. The parameters include: network bandwidth, cloud processing speed, device processing speed, the number of transferred bytes, and the energy consumption of a smartphone when it's in idle, processing and communicating states. However, the authors only discussed these various parameters and did not experiment their work on a real offloading system for real mobile applications.

Some previous work investigated the use of automatic program [58-61]. Cuervo et al. proposed MAUI [39], a system that enables energy-aware offloading of mobile code to the infrastructure. MAUI enables developers to produce an initial partitioning of their applications by annotating methods and/or classes as remotable. At runtime, the MAUI solver decides which remotable methods should execute locally and which should execute remotely. Unlike MAUI, Jade provides a sophisticated programming model with a full set of APIs, so developers have total control on: how application is partitioned, where code is offloaded and how remotable code interacts with local code.

Chun et al. proposed CloneCloud [40], an application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a

computational cloud. Compared with CloneCloud, code offloading in Jade is lightweight. Remotable objects are serialized, transferred, and deserialized, resulting in much lower overhead compared to thread migration.

4.7 Future Work

In our future work, we will focus on enhancing the capability of mobile devices with the help of cloud. Although mobile technology advances very fast, mobile devices are still not capable of handling complex computations. With computation offloading to cloud, mobile devices might be able to run complex applications. We will also study how cloud changes the design of mobile applications.

4.8 Conclusion

In this paper, we presented Jade, a system that enables computation offloading from mobile devices to the cloud. Jade can effectively reduce energy consumption of mobile devices, and dynamically change its offloading strategy according to device status.

We evaluated Jade with two applications: a face detection application and text search application. Results showed that Jade can effectively reduce energy consumption for both applications while improving their performance.

Chapter 5 - Conclusion

In this dissertation, I have presented a complete system – Jade, for reducing energy consumption of mobile devices. This work is significantly based upon the idea of computation offloading.

In chapter 1, we discussed one of the biggest challenges for mobile device – battery life. With the fast advancement of hardware technology, mobile devices are becoming more and more energy hungry. In contrast, there is no big breakthrough in battery technology. Balancing performance and battery life is a big challenge for mobile industry. One of the solutions to reduce energy consumption is computation offloading. We gave a review of computation offloading - its history, its goal and its applications. Researchers have been studying computation offloading, focusing on different areas, e.g., making offloading decisions, developing offloading infrastructures. Computation offloading can achieve two goals: shorten execution time and reduce energy consumption. In order to achieve these goals, the program needs to be partitioned into remotable parts which can be offloaded to remote server. Two different partition methods: static partitioning and dynamic partitioning are discussed. Mathematical models to guarantee the performance of computation offloading is also discussed. From these models we can see that not every type of code is suitable for computation offloading.

In chapter 2, we present a computation offloading system which can offload energy-intensive code from mobile device to servers. In order to optimize the energy savings, the system can automatically choose Wi-Fi and Bluetooth for data transfer based on the characteristics of applications. We show the architecture of the system and its implementation details. We evaluated the system with two applications.

In chapter 3, we present Jade which supports more devices, i.e., Android device, non-Android device and cloud platform. With the extensive support for devices, Jade becomes more useful for users under different scenarios: in the office, at home or traveling. We also implemented a new component in order to optimize the performance of servers (e.g., workload, battery level). We discussed a multi-level task scheduling algorithm which works on both the client and the server. The goal of the algorithm is to enhance the performance of application and balance the workload between servers. The experiments showed that such task scheduling strategy is important in a computation offloading system.

In chapter 4, we introduced an innovative improvement for Jade: separating the code and data. We analyzed the impact of data transfer on battery life, which shows that the size of data transferred over the network is the dominant factor affecting the performance of a computation offloading system. As the network interface of mobile device such as Wi-Fi is energy intensive, we need to reduce the amount of data transferred when computation offloading occurs. We noticed that most mobile devices need to be charged frequently in certain pattern (e.g., smartphone users often charge their devices at sleep time), during the charging period, data can be synchronized between client and server in advance before computation offloading happens. We showed the implementation of a component (MDSS) which performs such task. Evaluation showed that with MDSS, Jade only needs to offload app code to the server, thereby, further reducing the energy consumption for mobile devices.

Chapter 6 - Future Work

In our future work, we will study problems that are not addressed in our current system.

In this paper we assume that the environment in which we run Jade is trusted. We need to investigate which security measures are needed to secure the communication between the smartphone and the remote cloud resources. We also have to pay attention to the security implications of running foreign code on the remote servers, and validating whether the returned code is from the trusted source.

We will extend the programming model to support callbacks from the remote servers to the client and method parameters to be used as return values, like the AIDL specification supports.

We also want to add context information of the server to the system, such as processor speed, available battery, etc., which could lead to more sophisticated task scheduling algorithms and interesting scenarios. For example, if the client and server can switch roles depending on the battery level, then all connected devices will reach same battery level at some point and consume the battery with same pace. There must be some interesting applications which require system like this.

Nowadays, with the introduction and mass production of wearable devices like smart watch, extending Jade to support wearable devices is an interesting direction. Most wearable devices are small in size, making it difficult to put large battery into such devices. Thereby, battery life is even more sensitive and crucial for wearable devices. For example, most smart watches need to be charged everyday which is not convenient for users. We need to investigate the effectiveness of computation offloading on wearable devices. Since applications running on wearable devices like smart watch are different than these running on smartphones and tablets, we also need to investigate how to adopt these differences in Jade API without compromising its easy-to-use nature.

Bibliography

[1] Forman GH, Zahorjan J (1994) The challenges of mobile computing. Computer 27(4):38–47.

[2] Joseph AD, de Lespinasse AF, Tauber JA, Gifford DK, KaashoekMF (1995) Rover: a toolkit for mobile information access. In: ACM symposium on operating systems principles, pp 156–171.

[3] Kotz D, Gray R, Nog S, Rus D, Chawla S, Cybenko G (1997) Agent Tcl: targeting the needs of mobile computers. IEEE Internet Computing (4):58–67.

[4] Noble BD, Satyanarayanan M (1999) Experience with adaptive mobile applications in Odyssey. Mobile Netw Appl 4(4):245–254.

[5] Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B (1997) Concordia: an infrastructure for collaborating mobile agents. In: International workshop on mobile agents, pp 86–97.

[6] Wong D, Paciorek N, Moore D (1999) Java-based mobile agents. Commun ACM 42(3):92–102.

[7] White JE (1997) Mobile agents. In: Software agents (MITPress), pp 437–472.

[8] Qi M (1997) Resource conservation in a mobile transaction system. IEEE T Comput 46:3:299–311.

[9] Perkins CE (1996) Handling multimedia data for mobile computers. In: Computer software and applications conference, pp 147–148.

[10] Balan RK (2004) Powerful change part 2: reducing the power demands of mobile devices. IEEE Pervasive Comput 3(2):71–73.

[11] Chen G, Kang B-T, Kandemir M, Vijaykrishnan N, Irwin MJ, Chandramouli R (2004) Studying energy trade offs in offloading computation/compilation in java enabled mobile devices. IEEE Trans Parallel Distrib Syst 15(9): 795–809.

[12] Gu X, Nahrstedt K, Messer A, Greenberg I, Milojicic D (2003) Adaptive offloading inference for delivering applications in pervasive computing environments. In: IEEE international conference on pervasive computing and communications, pp 107–114.

[13] Gurun S, Krintz C (2003) Addressing the energy crisis in mobile computing with developing power aware software. Technical Report, Department of Computer Science, University of California, Santa Barbara.

[14] Gurun S, Krintz C, Wolski R (2004) NWSLite: a lightweight prediction utility for mobile devices. In: International conference on mobile systems, applications, and services, pp 2–11.

[15] Kremer U, Hicks J, Rehg J (2003) A compilation framework for power and energy management on mobile computers. In: Proceedings of the 14th international conference on Languages and compilers for parallel computing. Cumberland Falls, KY, USA, pp 115–131.

[16] Li Z, Wang C, Xu R (2001) Computation offloading to save energy on handheld devices: a partition scheme. In: International conference on compilers, architecture, and synthesis for embedded systems, pp 238–246.

[17] Li Z, Xu R (2002) Energy impact of secure computation on a handheld device. In: IEEE international workshop on workload characterization, pp 109–117.

[18] Rong P, Pedram M (2003) Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In: Conference on design automation, pp 906–911.

[19] Wang C, Li Z (2004) Parametric analysis for adaptive computation offloading. In: ACM SIGPLAN conference on programming language design and implementation, pp 119–130.

[20] Hong YJ, Kumar K, Lu YH (2009) Energy efficient content based image retrieval for mobile systems. In: International symposium on circuits and systems, pp 1673–1676.

[21] Nimmagadda Y, Kumar K, Lu Y-H, Lee CSG (2010) Realtime moving object recognition and tracking using computation offloading. In: IEEE international conference on intelligent robots and systems, pp 2449–2455.

[22] O'Hara KJ, Nathuji R, Raj H, Schwan K, Balch T (2006) Autopower: toward energy-aware software systems for distributed mobile robots. In: IEEE international conference on robotics and automation, pp 2757–2762.

[23] Wolski R, Gurun S, Krintz C, Nurmi D (2008) Using bandwidth data to make computation offloading decisions. In: IEEE international symposium on parallel and distributed processing, pp 1–8.

[24] Xian C, Lu Y-H, Li Z (2007) Adaptive computation offloading for energy conservation on battery-powered systems. In: International conference on parallel and distributed systems, pp 1–8.

[25] AnagnostouME, Juhola A, Sykas ED (2002) Context Aware services as a step to pervasive computing. In: Lobster workshop on location based services for

accelerating the European-wide deployment of services for the mobile user and worker, pp 4–5.

[26] Goyal S, Carter J (2004) A lightweight secure cyber foraging infrastructure for resource-constrained devices. In: Mobile computing systems and applications, pp 184–195.

[27] Guan T, Zaluska E, RoureD(2005)Agrid service infrastructure formobile devices. In: IEEE international conference on semantics, knowledge, and grid, pp 42–48.

[28] Guan T, Zaluska E, Roure D (2006) Extending pervasive devices with the semantic grid: a service infrastructure approach. In: IEEE conference on computer and information technology pp 113–118.

[29] Hemmes J, Poellabauer C, Thain D (2007) On-demand transient data storage and backup in mobile systems. In: IEEE military communications conference, pp 1–7.

[30] Yang K, Ou S, Chen H-H (2008) On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. IEEE communications magazine 46(1):56–63.

[31] Jamwal V, Iyer S (2006) Automated refactoring of objects for application partitioning. In: Asia-Pacific software engineering conference, pp 671–678.

[32] Ou S, Yang K, Hu L (2007) Cross: a combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments. In: IEEE global telecommunications conference, pp 720–725.

[33] Ou S, Yang K, Liotta A, Hu L (2007) Performance analysis of offloading systems in mobile wireless environments. In: IEEE international conference on communications, pp 1821–1806.

[34] Rim H, Kim S, Kim Y, Han H (2006) Transparent method offloading for slim execution. In: International symposium on wireless pervasive computing, pp 1–6.

[35] Seshasayee B, Nathuji R, Schwan K (2007) Energy aware mobile service overlays: cooperative dynamic power management in distributive systems. In: International conference on automatic computing, pp 6–12.

[36] Sivavakeesar S, Gonzalez OF, Pavlou G (2006) Service discovery strategies in ubiquitous communication environments. IEEE Commun Mag 44(9):106–113.

[37] Weinsberg Y, Dolev D, Wyckoff P, Anker T (2007) Accelerating distributed computing applications using a network offloading framework. In: IEEE international parallel and distributed processing symposium, pp 1–10.

[38] K. Kumar et al. A Survey of Computation Offloading for Mobile Systems. In Mobile Networks and Applications, volume 18, Issue 1, pp 129-140, 2012.

[39] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In MobiSys, 2010.

[40] B. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In ACM Eu-roSys, 2011.

[41] Trepn Profiler. https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler, 2014.

[42] S. Shankar, G. LaKomski, et al. Power-aware Work Stealing in Homogeneous Multicore Systems. FUTURE COMPUTING 2014, The Sixth International Conference on Future Computational Technologies and Applications. 2014.

[43] Highest Response Ratio Next. http://en.wikipedia.org/wiki/Highest_response_ratio_next, 2014.

[44] W. Jung, C. Wonwoo, et al. DevScope: a nonintrusive and online power analysis tool for smartphone hardware components. Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 2012.

[45] H. Qian and D. Andresen. Jade: An Efficient Energy-aware Computation Offloading System with Heterogeneous Network Interface Bonding for Ad-hoc Networked Mobile Devices. In Proceedings of the 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014.

[46] H. Qian and D. Andresen. Extending Mobile Device's Battery Life by Offloading Computation to Cloud. In Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2015.

[47] H. Qian and D. Andresen. Emerald: Enhanced Scientific Workflow Performance with Computation Offloading to the Cloud. In Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 2015.

[48] H. Qian and D. Andresen. An Energy-saving Task Scheduler for Mobile Devices. In Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 2015.

[49] H. Qian and D. Andresen. Reducing Mobile Device Energy Consumption with Computation Offloading. In Proceedings of 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015.

[50] H. Qian and D. Andresen. Jade: Reducing Energy Consumption of Android App. In the International Journal of Networked and Distributed Computing (IJNDC), Atlantis press, 2015.

[51] H. Qian and D. Andresen. Automate Scientific Workflow Execution between Local Cluster and Cloud. In the International Journal of Networked and Distributed Computing (IJNDC), Atlantis press, 2016.

[52] Q. Chen, H. Qian et al. BAVC: Classifying Benign Atomicity Violations via Machine Learning. In Advanced Materials Research, Vols 765-767, pp. 1576-1580, Sep, 2013.

[53] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-Grained Mobility in the Emerald System. ACM Transactions on Computer Systems, 6(1):109–133, 1988.

[54] A. Birrell, G. Nelson, S. Owicki, and E. Wobber. Network Objects. In Proc. of the Fourteenth ACM Symposium on Operatings Systems Principles (SOSP), 1993.

[55] L. Cardelli. A Language with Distributed Scope. In Proc. of the 22nd Symposium on Principles of Programming Languages (POPL), 1995.

[56] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A Toolkit for Mobile Information Access. In Proceedings of the Fifteenth Symposium on Operating Systems Principles (SOSP), 1995.

[57] R. S. Gray. Agent Tcl: a flexible and secure mobile-agent system. In TCLTK'96: Proceedings of the 4th USENIX Tcl/Tk Workshop, 1996, Monterey, CA, 1996.

[58] G. C. Hunt and M. L. Scott. The Coign Automatic Distributed Partitioning System. In Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA, February 1999.

[59] U. Kremer, J. Hicks, and J. M. Rehg. Compiler-Directed Remote Task Execution for Power Management. In Proceedings of The Workshop on Compilers and Operating Systems for Low Power (COLP), Philadelphia, PA, October 2000.

[60] M. Neubauer and P. Thiemann. From Sequential Programs to Multi-Tier Applications by Program Transformation. In Proc. of the Symposium on Principles of Programming Languages (POPL), Long Beach, CA, January 2005.

[61] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden. Wishbone: Profile-based Partitioning for Sensornet Applications. In Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI), Boston, MA, April 2009.

[62] Kumar.K,and Hsiang Lu.Y. 2010. Cloud computing for mobile users, Computer'99, volume 43, issue 4, 51-56.

[63] Miettinen.A.P, and Nurminen.J.K. 2010. Energy efficiency of mobile clients in cloud computing, 2nd USENIX conference on Hot topics in cloud computing, 4-4.

[64] Lin.K, Kansal.A, Lymberopoulos.D, and Zhao.F. 2010. Energy accuracy trade-off for continuous mobile device location. MobiSys'10, 285–298.

[65] Priyantha.B, Lymberopoulos.D, and Liu.J. 2011. LittleRock: Enabling Energy Efficient Continuous Sensing on Mobile Phones. IEEE Pervasive Computing Magazine, volume 10, issue 2, 12-15.

[66] http://www.seas.harvard.edu/news-events/press-releases/gates-grant/?searchterm=MFC. Last visited: March 29, 2013.

[67] American Chemical Society press. Mar 2011.http://portal.acs.org/portal/acs/corg/content?_nfpb=true&_pageLabel=PP_ART

ICLEMAIN&node_id=222&content_id=CNBP_026949&use_sec=true&sec_url_var=region1&__uuid=3d60a60f-1722-46c3-b6a3-d08cf4b7e46. Last visited: March 29, 2013.

[68] R. Bianchini and R. Rajamony, ”Power and Energy Management for Server Systems,” Computer, vol. 37, no. 11, pp. 68-74, 2004.

[69] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, “Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs”, in Proceedings of the ACM/IEEE SC 2005, Seattle, USA, November 2005.

[70] J.Markoff and S. Lohr, Intel’s huge bet turns iffy. New York Times Technology Section, September 29, 2002. Section 3, Page 1, Column 2.

[71] W. Feng, ”Making a Case for Efficient Supercomputing,” ACM Queue, vol. 1, no. 7, pp. 54-64, 2003.

[72] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, ”Power Optimization of Variable-Voltage Corebased Systems,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 12, pp. 1702-1714, 1999.

[73] T. D. Burd and R. W. Brodersen, ”Energy Efficient CMOS Microprocessor Design,” Proceedings of the 28th Annual Hawaii International Conference on System Sciences, pp. 288-297, January 1995.

[74] R. Ge, X. Feng, and K. W. Cameron, “Performanceconstrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters,” Proceedings of the ACM/IEEE SC 2005, Seattle, USA, November 2005.

[75] C. Hsu and W. Feng, ”A Power-Aware Run-Time System for High-Performance Computing,” Proceedings of the ACM/IEEE SC 2005, Seattle, USA, November 2005.

[76] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, ”Profile-based Optimization of Power Performance by using Dynamic Voltage Scaling on a PC cluster,” Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, April 2006.

[77] IBM Research Blue Gene Project. http://www.research.ibm.com/bluegene.

[78] N. D. Adiga, et al., ”An Overview of the BlueGene/L Supercomputer,” Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, USA, November 2002.

[79] W.Warren, E.Weigle, andW. Feng, “High-density Computing: A 240-node Beowulf in one cubic meter,” Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, USA, November 2002.

[80] Orion Multisystems. http://www.orionmult.com/.

[81] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, “Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs,” SC’06, November 2006.

[82] Zhang, L., Tiwana, B., Qian, Z., and Wang, Z. 2010. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and SystemSynthesis (Scottsdale, AZ, USA,

October 24 – 29, 2010). CODES+ISSS'10. ACM, New York, NY, 105-114. DOI= http://dx.doi.org/10.1145/1878961.1878982.

[83] Dong, M. and Zhong, L. 2011. 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 (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 335-348. DOI= http://dx.doi.org/10.1145/1999995.2000027.

[84] Dong, M. and Zhong, L. 2011. Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 85-98. DOI=http://dx.doi.org/10.1145/1999995.2000004.

[85] Anand, B., Thirugnanam, K., Sebastian, J., Kannan, P. G., Ananda, A. L., Chan, M. C., and Balan, R. K. 2011. Adaptive Display Power Management for Mobile Games. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (Washington, D.C., USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 57-70. DOI=http://dx.doi.org/10.1145/1999995.2000002.

[86] Pathak, A., Hu, Y. C., Zhang, M., Bahl, P., and Wang, Y. – M. 2011. Fine-grained Power Modeling for Smartphones Using System Call Tracing. In Proceedings of the ACM European Conference on Computer Systems (Salzburg, Austria, April 10 – 13, 2011). EuroSys'11. ACM, New York, NY, 153-167. DOI= http://dx.doi.org/10.1145/1966445.1966460.

[87] Carroll, A. and Heiser, G. 2010. An Analysis of Power Consumption in a Smartphone. In Proceedings of the 2010 USENIX Annual Technical Conference (Boston, MA, USA, June 23 – 25, 2010). USENIX ATC'10. USENIX Association, Berkeley, CA.

[88] Shye, A., Scholbrock, B. and Memik, G. 2009. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture (New York, NY, USA, December 12 – 16, 2009). MICRO'09. ACM, New York, NY, 168-178. DOI=http://dx.doi.org/10.1145/1669112.1669135.

[89] DS2784 Datasheet, http://pdfserv.maximic. com/en/ds/DS2784.pdf retrieved on 3/20/2012.

[90] Google Nexus One Specification, http://www.htc.com/us/support/nexus-one-google/tech-specs/retrieved on 3/20/2012.

[91] Sony Ericsson Xperia Arc, http://www.sonymobile.com/gb/products/phones/xperiaarc/specifications/ retrieved on 3/20/2012

[92] Monsoon Solutions, Inc., http://www.msoon.com/LabEquipment/PowerMonitor/ retrieved on 3/20/2012.

[93] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. 2010. Characterizing Radio Resource Allocation for 3G Networks. In Proceedings of the 10th ACM Internet Measurement Conference (New York, NY, USA, November 1 – 3, 2010). IMC'10. ACM, New York, NY, 137-150. DOI=http://dx.doi.org/10.1145/1879141.1879159.

[94] Haverinen, H., Siren, J. and Eronen, P. 2007. Energy Consumption of Always-On Applications in WCDMA Networks. In Proceedings of the IEEE 65th IEEE Vehicular Technology Conference (Dublin, Ireland, April 22 – 25, 2007). VTC'07. IEEE, 964-968. DOI= http://dx.doi.org/10.1109/VETECS.2007.207.

[95] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. 2011. Profiling Resource Usage for Mobile Applications: A Cross-layer Approach. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (Washington, D.C, USA, June 28 – July 1, 2010). MobiSys'11. ACM, New York, NY, 321-334. DOI= http://dx.doi.org/10.1145/1999995.2000026.

[96] Roy, A., Sumble, S., and Stutsman, R. 2011. Energy management in mobile devices with the Cinder operating system. In Proceedings of the ACM European Conference on Computer Systems (Salzburg, Austria, April 10 – 13, 2011). EuroSys'11. ACM, New York, NY, 139-152. DOI=http://dx.doi.org/10.1145/1966445.1966459.

[97] Vallina-Rodrigues, N. and Crowcroft, J. 2011. ErdOS: Achieving Energy Savings in Mobile OS. In Proceedings of the ACM 6th International Workshop on Mobility in the Evolving Internet (Washington, D.C., USA, June 28, 2011). MobiArch'11. ACM, New York, NY, 36-42. DOI= http://dx.doi.org/10.1145/1999916.1999926.

[98] Yoon, C., Kim, D., Jung, W., Kang, C. and Cha, H. 2012. AppScope: Application Energy Metering Framework for Android Smartphone using Kernel Activity Monitoring. In Proceedings of the 2012 USENIX Annual Technical Conference (Boston, MA, USA, June 13 – 15, 2012). USENIX ATC'12. USENIX Association, Berkeley, CA.

[99] AppScope, http://mobed.yonsei.ac.kr/~appscope/ retrieved on 7/25/2012.

[100] S. Zhang, X. Zhang, and X. Ou. “After We Knew It: Empirical Study and Modeling of Cost-effectiveness of Exploiting Prevalent Known Vulnerabilities Across IaaS Cloud”. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS 14), Kyoto, Japan, June, 2014.

[101] Su Zhang, Doina Caragea, and Xinming Ou. “An empirical study on using the National Vulnerability Database to predict software vulnerability”. In: Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA 11) , Toulouse, France, August 2011.

[102] Su Zhang, Xinming Ou, and John Homer. “Effective network vulnerability assessment through model abstraction”. In: Proceedings of the Eighth SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 11), Amsterdam, The Netherlands, July 2011.

[103] Yu, Jie, Sandra Skaff, Liang Peng, and Francisco Imai. "Leveraging Knowledge-based Inference for Material Classification." In Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, pp. 1243-1246. ACM, 2015.

[104] Fengguo Wei, Sankardas Roy, Xinming Ou, Robby. “Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps”. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014.

[105] Roelof Kemp, Nicholas Palmer. “Cuckoo: a Computation Offoading Framework for Smartphones”. In Proceedings of the International Conference on Mobile Computing, Applications, and Services (MobiCase), pp. 1-20, 2010.

[106] Eduardo Cuervoy, Aruna Balasubramanianz, Stefan Saroiux, Ranveer Chandrax, Paramvir Bahlx. “MAUI: Making Smartphones Last Longer with Code Offload”. In Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys), pp. 49 - 62, 2010.

[107] Galen C. Hunt and Michael L. Scott. “The Coign Automatic Distributed Partitioning System”. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 187-200, 1999.

[108] Eli Tilevich and Yannis Smaragdakis. “J-Orchestra: Enhancing Java Programs with Distribution Capabilities”. In ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 19, No. 1, Article 1, pp. 1-41, 2009.

[109] Michael Philippsen and Matthias Zenger. “JavaParty: Transparent Remote Objects in Java”. In Concurrency: Practice and Experience 9(11):1225-1242, John Wiley & Sons, Ltd., 1997.

[110] Alan Messer, Ira Greenberg, Philippe Bernadat, DeJan Milojieie, Deqing Chen, T.J. Giuli, Xiaohui Gu. “Towards a distributed platform for resource-constrained devices”. In Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 43-51, 2002.

[111] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. “Adaptive Offloading for Pervasive Computing”. In IEEE Pervasive Computing, pp. 66-73, 2004.

[112] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, et al. “CloneCloud: Elastic Execution between Mobile Device and Cloud”. In Proceedings of the European Conference on Computer Systems (EuroSys), pp. 301-314, 2011.

[113] Lei Wang and Michael Franz. “Automatic Partitioning of Object-Oriented Programs for Resource-Constrained Mobile Devices with Multiple Distribution Objectives”. In Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS), pp. 369-376, 2008.

[114] J. Flinn, S. Park, and M. Satyanarayanan. “Balancing Performance, Energy, and Quality in Pervasive Computing”. In Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 1-10, 2002.

[115] E. Lara, D. S. Wallach, and W. Zwaenepoel. “Puppeteer: Component-based Adaptation for Mobile Computing”. In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS), pp. 159-170, 2001.

[116] Eli Tilevich and Yannis Smaragdakis. “Portable and Efficient Distributed Threads for Java”. In Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware), pp. 478-492, 2004.

[117] Girvan, M. and NewMan, M.E.. “Community structure in social and biological networks”. In Proceedings of the National Academy of Sciences of the United States of America (PNAS), Vol. 99, pp. 7821-7826, 2002.

[118] Jonathan I. Maletic and Andrian Marcus. “Supporting Program Comprehension Using Semantic and Structural Information”. In Proceedings of the International Conference on Software Engineering (ICSE), pp. 103-112, 2001.

[119] Walter Binder, et al. “Using Bytecode Instruction Counting as Portable CPU Consumption Metric”. In Electronic Notes in Theoretical Computer Science, Elsevier, pp. 57-77, 2006.

[120] L. Zhang, B. Tiwana, Z. Qian, Z.Wang, et al. “Accurate online power estimation and automatic battery behavior based power model generation for smartphones”. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 105-114, 2010.

[121] Saleh E., Abdullahi, et al. “Garbage Collecting the Internet: A Survey of Distributed Garbage Collection”. In ACM Computing Surveys, Vol. 30, No. 3, pp. 330-373, 1998.

[122] © [2014] IEEE. Reprinted, with permission, from [Hao Qian, Jade: An Efficient Energy-aware Computation Offloading System with Heterogeneous Network Interface Bonding for Ad-hoc Net-worked Mobile Devices, July, 2014]

[123] © [2015] IEEE. Reprinted, with permission, from [Hao Qian, An energy-saving task scheduler for mobile devices, July, 2015]

[124] Peng, Liang, et al. "Highly accurate video object identification utilizing hint information." Computing, Networking and Communications (ICNC), 2014 International Conference on. IEEE, 2014.

[125] Wu, Huaming, Qiushi Wang, and Katinka Wolter. "Tradeoff between performance improvement and energy saving in mobile cloud offloading systems." Communications Workshops (ICC), 2013 IEEE International Conference on. IEEE, 2013.

[126] Wu, Huaming, Qiushi Wang, and Katinka Wolter. "Methods of cloud-path selection for offloading in mobile cloud computing systems." Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012.

[127] Wu, Huaming, Qiushi Wang, and Katinka Wolter. "Mobile healthcare systems with multi-cloud offloading." Mobile Data Management (MDM), 2013 IEEE 14th International Conference on. Vol. 2. IEEE, 2013.

[128] Wu, Huaming, and Katinka Wolter. "Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric." Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[129] Wu, Huaming, and Katinka Wolter. "Analysis of the Energy-Performance Tradeoff for Delayed Mobile Offloading", 2015.

[130] Wu, Huaming, Qiushi Wang, and Katinka Wolter. "Optimal Cloud-Path Selection in Mobile Cloud Offloading Systems Based on QoS Criteria." International Journal of Grid and High Performance Computing (IJGHPC) 5.4 (2013): 30-47.

[131] Qiushi Wang, Katinka Wolter. "Reducing Task Completion Time in Mobile Offloading Systems through Online Adaptive Local Restart", Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015.

[132] Qiushi Wang, et al. "Analysis of local re-execution in mobile offloading system", Proceedings of IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013.

Appendix

IEEE Thesis/Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.

2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.

3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]

2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.

3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is

permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org.er.lib.k-state.edu/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.