Accelerating Mobile-Cloud Computing Using A Cloudlet

by

Haoliang Wang

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Supervised by

Tolga Soyata

Department of Electrical and Computer Engineering
Arts, Sciences and Engineering
Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2013

Curriculum Vitae

The author was born in Shandong, P. R. China in 1989. Before he came to University of Rochester, he joined School of Physics and Optoelectronic Engineering at Dalian University of Technology in 2008, where he received a B.S. degree in Applied Physics in 2012. He began his graduate studies at the Department of Electrical and Computer Engineering at University of Rochester in 2012. He is currently working towards his M.S. degree in the area of mobile-cloud computing system under the direction of Professor Tolga Soyata.

Acknowledgments

I would like to thank many people who have helped me through the completion of this thesis.

First, I would like to express my deepest appreciation to my advisor and my committee chair, Professor Tolga Soyata, who guided my research and thesis throughout my graduate studies. His guidance, vision, funding as well as the rigorous attitude in research impress and inspire me all along. I would also like to thank Professor Wendi B. Heinzelman and Professor Mehmet Aktas for serving on my thesis committee.

In addition, I would like to thank other members in the MOCHA research group - Professor Minseok Kwon, Dr. Jiye Shi and my colleagues including Meng Zhu, Ba He, Zuochao Dou and Wei Liu for their suggestions and consistent support during the research. You have helped me overcome barriers in my research and provided an excellent and supportive environment to work in.

Finally, I would like to thank my parents, my girlfriend and friends. You have provided me with an overwhelming amount of support for which I am forever grateful.

Abstract

A mobile-cloud architecture provides a practical platform to perform compute-intensive operations, e.g. face recognition, on mobile devices. However, using a traditional mobile-cloud computing approach to perform real-time face recognition presents several challenges, including computation resource limitations, long network latencies high energy consumption issues and program partitioning burdens. To address the performance hurdles for mobile-cloud computing, in this paper, we determine three approaches - *preprocessing*, *caching* and *scheduling*, for accelerating compute-intensive applications by adding an intermediate resourceful device called *cloudlet*. We studied the structure and computational requirements of widely used face recognition algorithm thoroughly and implement the *Cloud Vision* system for mobile face recognition using the cloudlet to perform preprocessing and quantify the maximum attainable response time acceleration. Our results show up to 47% acceleration when appropriate mobile and cloudlet hardware is used in our system.

Contributors and Funding Sources

This work was supervised by a dissertation committee consisting of Professor Tolga Soyata and Professor Wendi B. Heinzelman of the Department of Electrical and Computer Engineering and Professor Mehmet Aktas from the School of Medicine and Dentistry at University of Rochester Medical Center.

The derivation of the server delay information were conducted by Zuochao Dou from Department of Electrical and Computer Engineering. The implementation of the Cloud Vision system was done together by the author and Meng Zhu from Department of Electrical and Computer Engineering. All other work conducted for the dissertation was completed by the author independently.

The work was supported in part by the National Science Foundation grant CNS-1239423 and in part by UCB Pharma and by CEIS, an Empire State Development designated Center for Advanced Technology, and by a gift from Nvidia Corporation.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

In past decades, significant advances in semiconductor technology have provided increasing computational power to the handheld mobile devices such as smartphones and tablets. The processing speed of such mobile devices has improve dramatically, providing the capabilities of not only accessing information anywhere and anytime, but also performing more sophisticated mobile augmented reality (MAR) processing in hand such as face and speech recognition, object detection and natural language processing (NLP). However, the potential of these applications is still constrained by relatively low computation capability, memory and storage of the mobile devices due to their limited size, weight and battery life. Therefore, applications which are both latency-sensitive and compute-intensive such as real-time face recognition are still beyond the capabilities of today's smartphones and tablets.

To overcome these limitations and extend the mobile devices' capabilities of supporting compute-intensive applications, mobile-cloud computing was introduced to leverage cloud resources, allowing mobile devices to utilize powerful cloud servers to process compute-intensive tasks on the backend. A mobile-cloud implementation of real-time compute-intensive applications, however, has a significant challenge: the high latency and limited bandwidth of the wide area net-

work (WAN) connection, which is unlikely to improve in the near future [2]. Taking the face recognition as an example, traditional mobile-cloud computing architecture requires mobile devices to send the entire image over the high-latency Internet, which will dramatically increase the application response time, contradicting the goal of real-time face recognition. Although performing preprocessing (face detection) on the mobile devices and sending only the preprocessed intermediate data (faces) to cloud servers may decrease the communication cost, further investigation shows that this approach has serious performance and energy issues under complex and demanding situations even with dedicated hardware for acceleration.

Recently, a mobile-cloudlet-cloud architecture called MOCHA (MObile Cloud Hybrid Architecture) has been developed at University of Rochester as a framework for running compute-intensive mobile applications with real-time latency requirements [3–5]. The introduction of a *cloudlet*, which is resource-rich device located nearby to the mobile devices and serves as an edge-server via low latency local area networks, provides the potential to address the performance and energy hurdles for mobile-cloud computing mentioned above.

The goal of this thesis is to explore the benefits in utilizing a cloudlet in the infrastructure to accelerate the performance of mobile-cloud computing. A prototype of mobile face detection and recognition system, as an example of our target applications which are both compute-intensive and latency-sensitive, is investigated, implemented and evaluated in terms of response time under MOCHA architecture. The goal of this thesis does not include the developing new approaches for code partitioning or offloading or implementing a mobile-cloud computing face recognition platform that is fully optimized and ready for real-world deployment.

1.2 Contribution

The main contribution of this thesis is the demonstration of the performance benefits in utilizing a cloudlet to accelerate mobile-cloud computing. We introduce a mobile-cloudlet-cloud architecture and analyze the potential performance benefits gained by utilizing the cloudlet as an accelerator. A range of work has been done to achieve this goal.

1. We study the mobile-cloudlet-cloud architecture in deep and determine three approaches for the cloudlet to improve the performance of mobile-cloud computing: *Preprocessing*, *Caching* and *Scheduling*.

2. The algorithm structure of face recognition is investigated. Based on its computational requirements, the recognition operation is partitioned into three distinct steps - *Face Detection*, *Projection* and *Database Search* to adapt the mobile-cloudlet-cloud architecture.

3. To examine the our ideas, a real-time face recognition system - *Cloud Vision* is implemented using OpenCV library[1]. We develop C code on cloudlet and cloud servers and Java code on Android mobile devices. Socket protocol is used for communication and the Internet latency is simulated based on real measurement from PlanetLab[2].

4. Our implementation achieves a 200x reduction in the size of the uploaded data, from entire images (1280x720, 200kB each), to eigenvectors (less than 1kB). Experiments with three simulated cloud server within U.S lead to an acceleration (in terms of response time) of up to 47% compared with

[1]OpenCV Library (Open Source Computer Vision Library) [6] is an open source BSD-licensed library that includes several hundreds of computer vision algorithms including face recognition.

[2]PlanetLab [7] is a group of computers located around the world, available as a testbed for computer networking and distributed systems research.

traditional mobile-cloud architecture. Based on the experiment result, the proper hardware configuration for a cloudlet to achieve the performance goal is also determined.

Our results show:

Utilizing a cloudlet in mobile-cloud computing provides significant performance accelerations (in terms of response time) by offloading preprocessing to the cloudlet.

1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide background information on the state of the art of the mobile-cloud computing and brief introduction on the target applications which MOCHA architecture is designed to accelerate. In Chapter 3, we provide the detailed analysis of the capabilities of the cloudlet as an accelerator and its potential improvements for the overall mobile-cloud computing performance. In Chapter 4, we introduce the algorithm structure and the system architecture of the proposed Cloud Vision system. Our experimental setup and result are showed in Chapter 5, followed by the discussion and future work in Chapter 6. Our final conclusion is provided in Chapter 7.

2 Background

In this chapter, background information about state of the art of the mobile-cloud computing and our targeting applications are provided.

2.1 Mobile-Cloud Computing

With the explosion of mobile applications and the support of cloud computing for a variety of services for mobile users, mobile-cloud computing is introduced and intensively investigated as an integration of cloud computing into the mobile environment [2, 4, 5, 8–15]. Mobile-cloud computing brings new types of services and facilities for mobile users to take full advantages of cloud computing.

The primitive form of mobile-cloud computing such as Gmail for iOS is an application running on the mobile devices, requesting services on a remote resource rich server via Wi-Fi and/or 3G connections using the existing client-server communication framework like XML-RPC[1]. Based on this simple model, improvements have been made by researchers by using *computation offloading* techniques. Besides, with the rapid development of the computation capability on recent mobile devices, they are now able to become resource providers, collectively forming a local *mobile cloud platform* which can be both energy efficient and relatively

[1]XML-RPC is a remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism [16].

powerful. The state of the art of these two research area will be discussed in the rest of this section.

2.1.1 Computation Offloading

Offloading is a solution to alleviate resource limitations on the mobile devices and provide more capabilities for these devices by migrating partial or full computations (code, status and data) to more resourceful computers [17]. The rapid development of wireless network connectivity and mobile devices in recent years has enabled the feasibility of computation offloading. Currently, most of the computation offloading researches are focusing on the following aspects.

- *What to offload.* Not the entire program can be offloaded for remote execution. Before offloading, the program needs to be partitioned manually by the programmer or automatically by the compiler or runtime. Manual partitioning will put a burden on the programmers but will have lower overhead. On the contrary, the automated partitioning can perform offloading on a unmodified program but with a higher overhead. Different strategies like code tagging and dynamical prediction based on profiling can be applied to increase the performance.

- *When to offload.* Applications may have different requirements on performance and mobile devices may have different capabilities and energy concerns. Offloading decisions need to be made based on targets of improving performance and/or saving energy as well as the network conditions. These decisions can be made by statically and/or dynamically profiling, which has impact on execution overhead.

- *How to offload.* The development of virtualization and the emerging cloud computing technologies provides a powerful, flexible, manageable and secure

platform for offloading, attracting significant researches on VM (Virtual Machine)-based offloading approaches. The granularity varies from OS-level, application/thread-level to method-level.

Three computation offloading system with different design focuses - Kimberley, CloneCloud and MAUI are briefly introduced below.

OS-level offloading

To achieve the goal of both performance and manageability, the VM-based Kimberley architecture is proposed by Satyanarayanan et al [2]. A cloudlet defined as a self-managed *datacenter in a box* is introduced in Kimberley. The cloudlet is able to support few users at a time and maintains only soft state hence the loss of connection is acceptable.

When a mobile client connects to the cloudlet, it notify the Kimberley Control Manager (KCM) on the cloudlet to download a small VM overlay, which is generated by comparing the target customized VM image with the base VM, from either Internet or the mobile client. When the VM overlay is delivered, a technique called dynamic VM synthesis will transiently creates and launches the target VM. After the computation is done, the KCM can simply shutdown the VM and free the resources, providing a self-manageability that only needs minimal maintenance.

The Kimberly system is implemented on a Nokia N810 tablet running Maemo 4.0, and the cloudlet infrastructure was implemented on a desktop computer running Ubuntu where VirtualBox is used to provide VM support. System performance is evaluated by VM overlay sizes and the speed of the synthesis. The VM overlay sizes were 100-200 MB for a collection of Linux applications, an order of magnitude smaller than a full VM image size (8 GB). The speed of synthesis ranged from 60 to 90 seconds which has plenty of room for improvement through

further optimization like parallelized compression and decompression and VM overlay prefetching.

The strengths of Kimberley is the self-manageability of the cloudlet and high flexibility for programmers to configure the code on the cloudlet since they have full control of OS on isolated VMs. The weaknesses of Kimberley design is that the programmer needs to decide what to offload and manually partition the program and the huge initialization overhead.

Thread-level offloading

In order to free the programmer from manual program partitioning for offloading, Chun et al. proposed the CloneCloud system, allowing unmodified program to be accelerated by offloading portion of the execution at a granularity of thread [10].

To achieve this, they modified the Dalvik VM[2]. The modified runtime will rewrite the executable of user's program and insert migration points by statical analysis. When the program is running, at those automatically chosen points, individual threads will migrate from the mobile device to a device clone in the cloud, and the User Interface (UI) or other essential components can remain to be executed at the mobile but will block if accessing status of migrated threads. A dynamic profiler is used to modeling of execution, migration and energy cost of each method on a mobile device, and then an optimization solver is used to decide the migration points based on the given optimization objectives.

The Android-based CloneCloud system prototype is implemented on an HTC G1 mobile phone and a server running the Android x86 virtual machine via VMware ESX 4.1 where the mobile clones are running. Three applications a virus scanner, image search, and privacy preserving targeted advertising were

[2]Dalvik VM is an open process-level virtual machine used in Android which is a Google-backed open source Linux-based mobile operating system.

tested on the CloneCloud prototype. The results show that for the tested applications, when connecting to the CloneCloud via Wi-Fi, the execution time is shortened by 2.1x-20x and the energy consumption is reduced by 1.7x-20x. When connecting to the CloneCloud via 3G, the execution time is shortened by 1.2x-16x and the energy consumption is reduced by 0.8x-14x.

The strength of the CloneCloud system is that it achieves distributed execution without manually modifying the source code, taking the programmer out of the burden for program partitioning. The weakness is that for complex application, the overhead to transfer the state (heap and stack) may counterweight the performance gain and energy savings from offloading. Besides, the security issues are not considered in the CloneCloud system.

Method-level offloading

Motivated by the fact that the energy consumption will remain the primary bottleneck for handheld mobile devices, MAUI (Mobile Assistance Using Infrastructure) is proposed to address the problem by maximizing energy saving by computation offloading [9]. They discovered that the completely automated program partitioning and coarse-grained offloading will increase the overhead, consuming more energy. To decrease the overhead while minimizing the programmer's burden, they use more fine-grained method-level offloading and the target method is identified by programmers' annotations in the source code.

MAUI is built on Microsoft .NET Common Language Runtime (CLR) for code portability. The programmer decide which methods may be offloaded and annotate them with tags. Those methods along with the necessary program state will be extracted using reflection and type-safety. Then the MAUI profiler will profile each method and use serialization to determine the offloading costs. Combining measurements of processing and transferring, a MAUI solver will decide whether

the method is worth offloading based on the solution to a linear programming formulation. MAUI generates two proxies on both the mobile device and the server that handle control and data transfer. The MAUI coordinator on the server side will handle the authentications, resource allocations and executions.

The mobile part of MAUI was implemented on an HTC Fuze mobile phone running Windows Mobile 6.5 with the .NET Compact Framework v3.5, and the MAUI server was implemented on a desktop running Windows 7 with the .NET Framework v3.5. The main results measure energy consumption and execution time for three applications face recognition, 400 frames of a video game, and 30 moves in a chess game. The results show that using remote execution on MAUI saves 5x-12x the energy compared to the mobile-phone-only case and reduces the execution time by more than a factor of 6.

2.1.2 Mobile Cloud Platform

The word *Cloud* is usually considered to be a collection powerful servers. However, with increasing processing capabilities on the mobile devices, a collection of such mobile devices connected via a local ad-hoc network is now powerful enough to be considered as a mobile cloud to provide computation resources just like the conventional servers. Recently, mobile cloud has been investigated as a powerful and more importantly,an energy-efficient platform to support compute-intensive and parallelizable applications. Examples of using mobile devices as a cloud of computing resources - Hyrax, NativeBOINC and GEMCloud, will be described below.

Hyrax

Apache Hadoop [18] is an open-source implementation of the MapReduce programming model. It is originally designed to run on a cluster of powerful servers.

To utilizing the mobile devices as the computation unit, Marinelli ported Hadoop to Android platforms and proposed the Hyrax system [19]. Hyrax enables computation jobs to be executed on distributed mobile devices connected by wireless network.

A distributed multimedia search and sharing application is implemented on Hyrax. Experiments shows that Hyrax can easily scale up to 10 HTC G1 and 5 HTC Magic mobile phones running Android 1.5 in terms of execution time and resource usage. The energy efficiency of Hyrax is showed to be significantly high than traditional server clusters. However, the performance of Hyrax was poor compared with Hadoop on traditional servers, not only because the mobile devices (ARM11-based CPU of 528MHz) and the wireless network (802.11g wireless router with a maximum bandwidth of 54 Mbps) were not powerful enough, but also because Hadoop was not originally designed, nor optimized, for mobile devices, causing huge overhead within the system.

NativeBOINC

The NativeBOINC [20] is an Android implementation of the BOINC (Berkeley Open Infrastructure for Network Computing) [21] which is an open-source volunteer computing software utilizing crowd-sourcing for scientific computing. NativeBOINC for Android allows mobile device users to choose projects, start and stop them on demand, contributing their free computing power. Experiment [22] shows that the ARM-based mobile processors have energy efficiency advantages over the traditional Intel desktop processors.

GEMCloud

GEMCloud (Green Energy Mobile Cloud) is another example of using mobile devices to create an ad hoc cloud of computing resources [23]. By utilizing dis-

tributed mobile devices to cooperatively accomplish large parallelizable computational tasks, the author envisions that such approaches can make use of the massive amount of idle computing power that is potentially available to the public. More importantly, the authors show that a mobile computing system like GEMCloud has significant advantages in energy efficiency over traditional desktop cloud servers when the overall system is considered, rather than each individual computational device.

2.2 Mobile Augmented Reality

Augmented reality (AR) is a live, direct or indirect, view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data [24]. With the increasing performance and popularity of mobile devices, mobile augmented reality (MAR), as more natural way instead of using desktops and workstations, has gained significant interests [25]. As mentioned in the previous chapter, to provide satisfying user experience, MAR usually requires real-time responsivity and intensive computation which either mobile devices alone or mobile-cloud computing are not capable of. In this section, several possible applications which our proposed system is targeting to accelerate will be briefly introduced.

2.2.1 Object Detection and Recognition

Object detection is to find a series of objects with common attributes (face, human, tank, etc) from an image or video sequences. After a series of objects are detected, we need to recognize them, usually within a database. Intensive researches in computer vision have made object detection and recognition feasible, enabling applications in the following areas.

Battlefield

Through the assistance of real-time object detection and recognition from wearable devices, the soldiers can identify threats or friendlies more accurately and efficiently, significantly lowering the potential casualties.

Surveillance

At public places like the airport or the president giving a speech, where security is critical, real-time face recognition and action detection can assist the agents to find the potential terrorists hidden inside the crowd.

Archeology

Object detections can be used to aid archaeological research, by detecting and augmenting archaeological features onto the modern landscape, enabling archaeologists to formulate conclusions about site placement and configuration.

2.2.2 Speech and Speaker Recognition

Speech recognition is to translate the incoming voice data into human understandable text for reading or further processing. Voice translation is a typical application for speech recognition. Speaker recognition is a way to use biometrics to recognize the identity of the speaker from a piece of voice for verification or identification purposes.

3 Utilizing a Cloudlet as an Accelerator

Although mobile devices have been improved dramatically over the past few years, they are still relatively limited in processing speed, memory, storage, battery life and network bandwidth. For latency-sensitive and compute-intensive applications, it is important to reduce the application response time to provide the best user experience. Because of the inconsistent network conditions over the Internet and the possible unavailability of cloud servers, a cloudlet can be introduced to provide local computing power and storage and the intelligence for task management. Figure 3.1 shows an example of a mobile-cloud architecture that utilizes a cloudlet as a local edge server that can communicate with the mobile over a local area network (LAN).

Here in this paper, we define a cloudlet as follows:

- A resourceful device that has relatively powerful CPU, GPU and high internal storage which can support requests from tens of mobile clients.
- A nearby device that communicates with mobile devices via single-hop high-speed connections such as Wi-Fi. Since large amount of data need to be transfered between the mobile devices and the cloudlet, low-speed multi-hop connections via the WAN will counterweight the performance gains.
- A always-on devices which is equipped with constant power supply so that

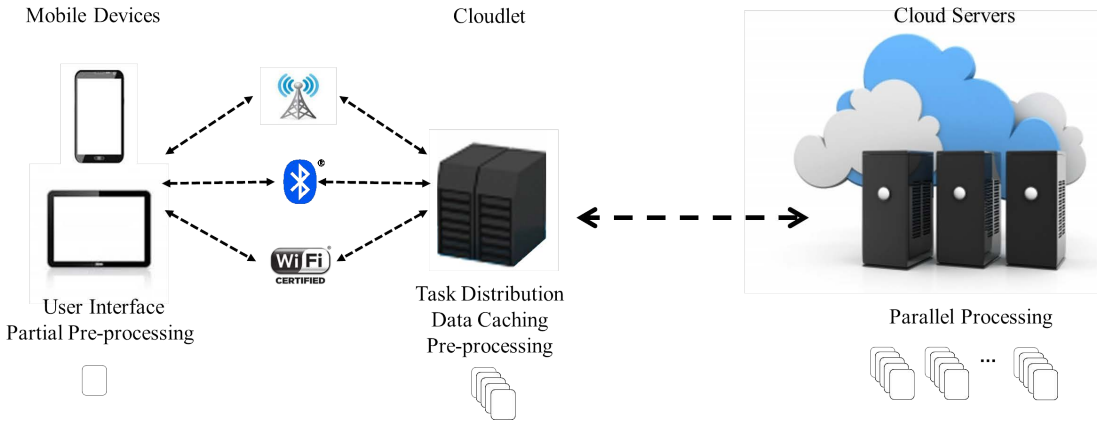

Figure 3.1: Acceleration can be achieved by utilizing the cloudlet as a computation and communication buffer.

we can offload computations to the cloudlet to save energy for the mobile devices.

Though similar device has been proposed in other papers [2,12], the capability of a cloudlet to accelerate mobile-cloud computing is still unclear. In this paper, we have summarized three specific ways that a cloudlet may help to reduce application response time and improve the overall performance:

- Preprocessing
- Caching
- Scheduling

These there approaches will be described in detail in the rest of this chapter.

3.1 Preprocessing

The cloudlet can use its better computing capability to perform preprocessing and reduce the size of data that must be transmitted to the cloud via the Internet, so

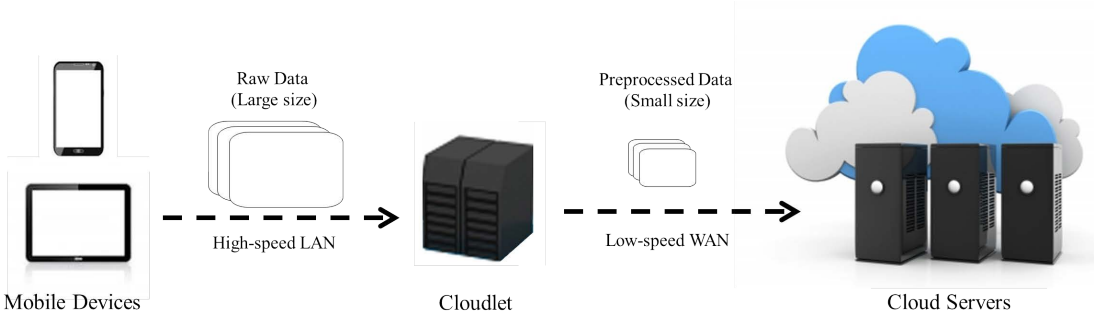

Figure 3.2: Illustration of preprocessing on the cloudlet.

that the latency for data transmission can be reduced.

To offload the computation to the cloud, mobile devices usually have to transmit large amount of raw data over the Internet, which will dramatically increase the latency, cause the application fail to reach the real-time goal. Preprocessing, from simple compression to highly sophisticated operations, can be done to the raw data to reduce its size. However, due to the limited computation power and battery life on the mobile devices, preprocessing on mobile devices will be both slow and power consuming. The additional computation latency may counterweight the benefits, or even make the overall latency worse.

As shown in Figure 3.2, by adding a cloudlet, the mobile devices now can offload the preprocessing tasks to the cloudlet via high-speed Wi-Fi connection. The cloudlet is able to significantly accelerate the preprocessing with its powerful processor and sends the preprocessed intermediate result to the cloud servers. The latency over the Internet is therefore greatly reduced and the additional latency of both the hop between mobile devices and the cloudlet and the processing on the cloudlet is small or even negligible, resulting a significant improvement of the overall latency.

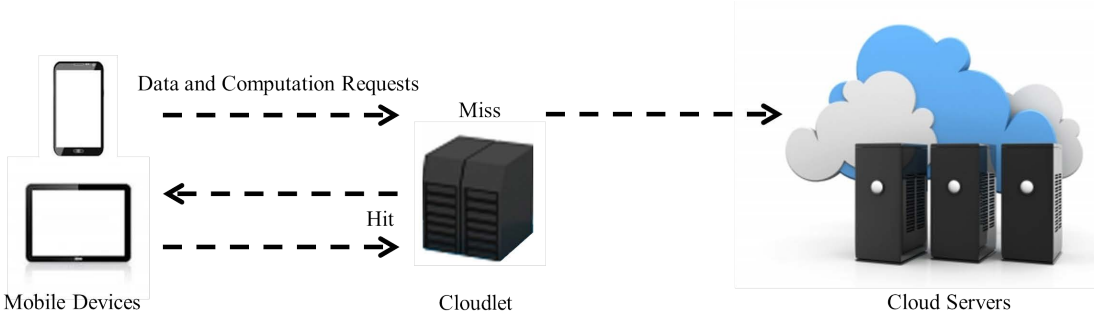

Figure 3.3: Illustration of caching data and computation on the cloudlet.

3.2 Caching

The cloudlet can utilize its highly available internal storage to cache a portion of the database from the cloud so that appropriate data can be delivered over the local network to the mobile when it needs this data as shown in Figure 3.3.

Most of the target applications are locations-related and the cloudlet is designed for serving mobile devices through local area network. Therefore, it is possible for using a cloudlet to enable fast data sharing and collaborations within nearby mobile devices. For an example of multiple mobile devices in the nearby locations performing face recognition, it is highly possible that the recognized face in the one mobile device will be captured by other devices later. Instead of sending redundant recognition requests to the cloud servers, with the presence of a cloudlet, the recognized result can be cached in the cloudlet and provided through local network when requests from others match or "hit" the result. Without en routing requests to the cloud, the high latency over the Internet can be eliminated and the loading on the cloud servers can be significantly filtered, providing the potential for the cloud servers to serve more mobile devices at the same time.

3.3 Scheduling

The cloudlet has the ability to scheduling resources of multiple cloud servers to serve multiple mobile devices. The cloudlet can provide profiling of all the available resources in the network and then perform intelligent task distribution and therefore optimize the overall performance and ensure the Quality of Service (QoS). Sophisticated algorithms that model the cloud-to-cloudlet delays as a set of graph edges/vertices [26–31] and use a set of integer linear inequalities can be incorporated when local computational resources ar available that permit the solution of computationally-intensive Integer-Linear Programming (ILP) algorithms.

In traditional mobile-cloud computing architecture, when mobile devices offload computation to the cloud, there are usually multiple available cloud servers with different network and loading conditions. Instead of choosing a fixed server or choosing the server randomly, the mobile device should choose a server that can offer the lowest possible latency for the offloading task. The mobile devices can keep tracking the servers status by constant profiling, reducing the battery life and causing redundant requests and network congestions when there are multiple mobile devices. Besides, since there is no coordination between mobile devices, it is possible that several mobile devices all en route their requests to one single server but leave other servers idle. This underutilization of the cloud server resources will result in poor performance.

With the presence of a cloudlet, multiple mobile devices can share the profiling result provided by the cloudlet, reducing the profiling requests to the cloud servers and energy consumption on the mobile devices. Besides, as a coordinator, the cloudlet is aware of multiple mobile devices and cloud servers, based on QoS requirements, several scheduling strategies can be applied by the cloudlet to increase the overall throughout and reduce the latency of a single task.

Fairness Scheduling to Maximize Throughput

For a given tolerable latency limit, the throughput can be maximized by scheduling the tasks fairly to each cloud servers according to their processing speed, current loading and the network latency for transferring the data so that as many tasks as possible can be done within their latency limits. Greedy algorithm can be used to decide which server should the task be scheduled to - cloud servers with high processing speed, low current loading and high speed network connectivity should process more tasks or vice versa.

Besides the fairness at the server side, cloudlet can schedule the tasks in a such way that the fairness in the client side can be achieved to avoid starvation. Prioritized scheduling can also be applied to serve important mobile clients first.

Parallelization and Redundant Scheduling to Minimize Latency

Assuming the task is parallelizable, the cloudlet can schedule the task to multiple cloud servers to reduce the processing and transmitting latency. Considering the instability of the network and the servers' loading conditions, the gain of parallelization may be reduced by the laggard. To alleviate the effect of such instabilities, redundant tasks can be scheduled to multiple servers [32]. The task is considered done when the first result comes back and other redundant tasks will be either ignored or aborted. The process is shown in Figure 3.4.

In this paper, we introduce a mobile-cloudlet-cloud-based face recognition system - *Cloud Vision* that allows multiple mobile devices to perform near-real-time face recognition by taking advantage of the cloudlet and multiple cloud servers. By utilizing a cloudlet as a preprocessing unit, the raw image can be reduced from hundreds of kilobytes to hundreds of bytes, thereby reducing the communication latency at the expense of computation. Potentially, the cloudlet can cache a portion of the face database, which can be permanently stored only in the cloud,

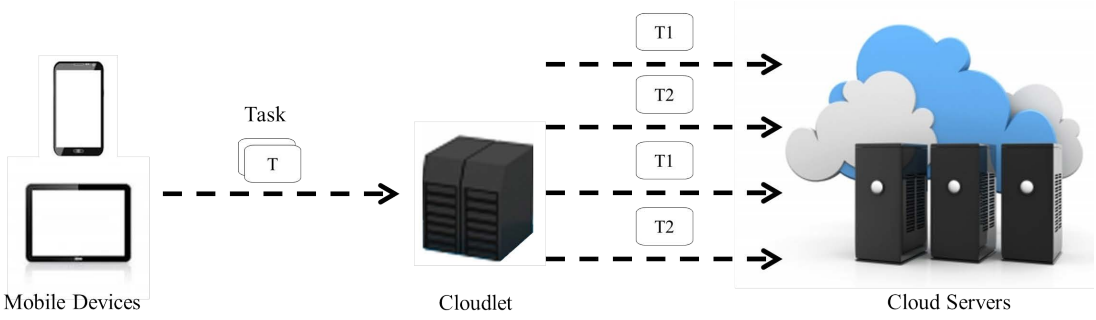

Figure 3.4: Illustration of parallelization and redundant scheduling on the cloudlet.

and thereby reducing the burden of the mobile-cloud communication as well as improving the overall system performance. Furthermore, the cloudlet is able to serve multiple mobile phones at the same time and is able to optimize their performance by applying a scheduling algorithm. In this paper, we will focus on the analysis of the benefits of using a cloudlet for preprocessing and scheduling for a face recognition application.

4 The Cloud Vision System

In previous sections, we mentioned that mobile-cloud-based face recognition is one of the applications that can benefit from the existence of a cloudlet. In this section, the proposed Cloud Vision system will be introduced.

The main design focus of the Cloud Vision system is solely the performance. Therefore, instead of the popular VM-based offloading approaches, the traditional client-server communication framework is used to reduce the overhead. Meanwhile, the program is manually pre-partitioned based on the algorithm structure. The face detection and recognition algorithms we used and their computational requirements will be studies in detail. System components and the partitioning decisions are then analyzed.

4.1 Algorithm Structure

The algorithmic structure of the face recognition application contains three distinct steps: face detection (**FD**), conversion of the detected face to a set of coefficients, forming an eigenvector (an operation defined as projection in the literature, **PJ**) and a database search (**S**). Although the last two steps (i.e., PJ and S) belong to the same face recognition algorithm, we study them separately in this paper, since they may be executed in different devices (e.g., PJ in the cloudlet and S in the cloud). In this section, these three separate steps are introduced in detail.

Though we choose to implement face detection and recognition, our system can be easily modified to support other similar applications. These application, though have different algorithms, can be partitioned and adapted to our Cloud Vision system the same as the face recognition.

4.1.1 Face Detection

For further processing, useful information needs to be isolated from the raw input data, that is face detection. Proposed by Viola and Jones in 2001 [33], the Viola-Jones object detection framework is used in this paper. This is the first object detection approach that provides competitive detection rates in real-time. In their object detection framework, three main techniques are used to dramatically accelerate the detection speed: using integral images and Haar-like features, using an AdaBoost classifier and using a cascade architecture for the AdaBoost classifier. Using Haar-like features from an integral image, the computation time of feature extraction is significantly reduced. Meanwhile, by organizing many weak classifiers in a cascade architecture in order of their complexity, non-objects can be rejected as soon as possible, avoiding extra useless evaluations and therefore offering higher performance. This algorithm is highly correlated and is difficult to divide into distinct steps therefore the face detection operation is treated as a single step in this system.

4.1.2 Face Recognition

After the face is detected, we can use the eigenface approach [34] to recognize this face within the database. It is considered the first successful example of facial recognition technology. It seeks to implement a system capable of efficient, simple, and accurate face recognition in a constrained environment. The system does not depend on 3-D models or intuitive knowledge of the structure of the face (eyes,

nose, mouth). Classification is instead performed using a linear combination of characteristic features (eigenfaces).

In this approach, the detected face is projected onto the eigenface space to generate the eigenvector, which makes comparisons with potential faces much easier. The eigenfaces are generated from a database by using Principle Component Analysis (PCA) to extract a low-dimension representation from the higher-dimension image space (i.e., the database). Based on this theory, the recognition process can be divided into two steps: projection and database search.

Initialization

Before recognition, the algorithm has to initialize. The initialization is to perform PCA on a database consisting of human face images to generate a set of eigenfaces: project all faces in the database onto a linear space formed by those eigenfaces as a set of basis functions and store the corresponding coefficient vectors called eigenvectors that uniquely represent each face. In theory, the maximum number of all possible eigenfaces equals to the number of images in the database and the more eigenfaces are used, the more accurate the prediction will be. However, a large number of eigenfaces will lead to long processing time. Based on the mathematical theory of PCA, we can ignore a large portion of the eigenfaces without losing much precision by keeping those with the largest eigenvalues. The eigenvalue associated with each eigenface represents how much an image in the database varies from the mean image in that direction. The eigenfaces with small eigenvalues only contain information about detailed differences, which is not critical under most cases. Figure 4.1 shows the relationship between the number of eigenfaces chosen and the percentage of information of the entire database covered. The percentage of information covered by the first k eigenfaces $P(k)$ is calculated as shown in

equation 4.1,

$$P(k) = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} \Omega_{i,j} \right)^2 / \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \Omega_{i,j} \right)^2 \tag{4.1}$$

where $\Omega_{i,j}$ is the i^{th} component (eigenvalue) of the eigenvector for the j^{th} image in the database and n is the total number of images in the database. As can be seen in Figure 4.1, a small number of eigenfaces can cover most of the information in the database. Another interesting observation here is that the smaller a database size is, the higher ratio of eigenfaces is necessary to cover the same percentage of information. However, for smaller databases, to counter this effect, one can take advantages of the fact that there are fewer points in the space formed by the eigenfaces, reducing the average distance necessary to distinguish any two faces. This provides a heuristic principle of how to determine the number of eigenfaces. In our system, we choose the number of eigenfaces based on this principle - 8x smaller than the number of total images in the database.

Projection

The next step is projection. An input face will be projected onto the same space formed by the eigenfaces, and a corresponding eigenvector will be produced for further processing. The projection operation is essentially a matrix multiplication operation and its computation complexity is related to the number of eigenfaces. Comparing a 4KB face image with its projected eigenvector, which is 4B multiplied by the number of eigenfaces, we can expect a large amount of savings as long as the database is small (in our case a data base of eight thousand images is the threshold).

Database Search

The last step of FR is to use nearest neighbor searching (NNS) to find any matches between vectors in the database and the input vector. We use Euclidean distance

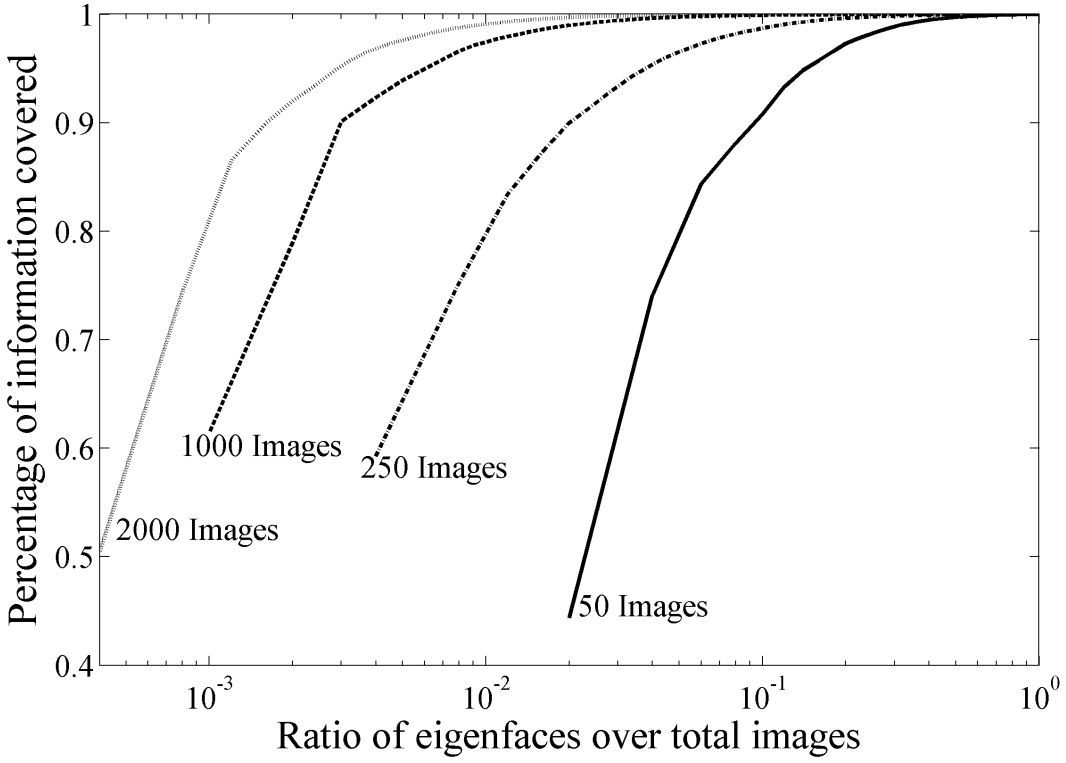

Figure 4.1: The relationship between the ratio of eigenfaces (number of eigenfaces divided by the total number of images in the database) and the percentage of information covered by these eigenfaces.

here. The distance between the test face and the k^{th} face in the database $d_E(k)$ is shown in equation 4.2,

$$d_E(k) = \sqrt{\sum_{i=1}^{n} (\Omega_{i,testface} - \Omega_{i,k})^2} \tag{4.2}$$

where Ω^i is the i^{th} component of the eigenvector for the k^{th} face in the database and n is the total number of eigenfaces. The face whose eigenvector yields the nearest distance will be the prediction of the input face.

4.2 System Architecture

In this section, based on the above analysis of the algorithm structure, we will describe our system components and determine which part of the algorithm should be execute on mobile devices, cloudlet or cloud servers.

4.2.1 Mobile

In traditional mobile-cloud architecture, mobile devices send the entire raw image to the remote sever which performs all the operation and send the recognition result back. This approach, being straightforward, will put a lot of burden on the network bandwidth and increase the response time of the application. For example, the size of a frame with a resolution of 1280x720 is around 200KB, and hence sending at 7fps will lead to a data rate of 11Mb/s. Given the asymmetric properties of the high speed Internet connection, this uploading rate can easily saturate the bandwidth, not to mention there could be multiple phones sending at the same time. Also, from Figure 4.2 obtained from PlanetLab, we can see that both the mean of the communication delay between cloud servers and mobile devices and its instability will increase linearly with the data size. Uploading

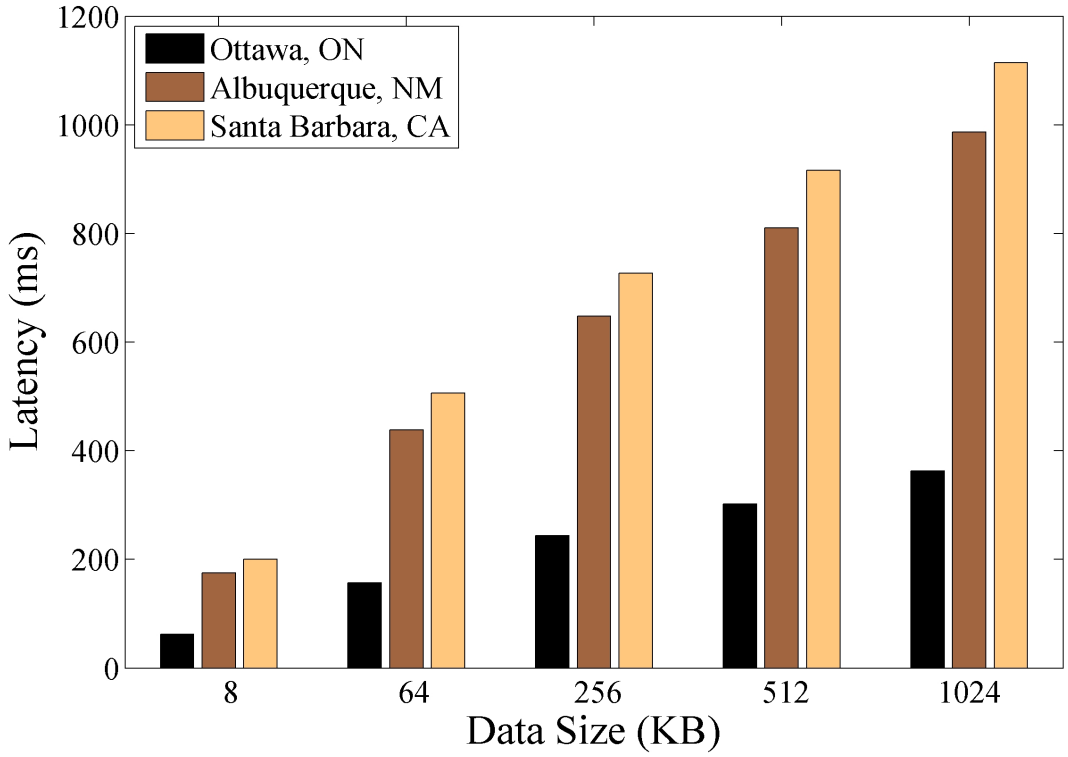

Figure 4.2: The mean communication latencies as the data size changes for communication between three Planet-Lab nodes and our node in Rochester, NY [1].

a large amount of data will lead to long response latency, which would not be tolerable for augmented reality applications such as face recognition.

To optimize the performance, currently face detection is usually done by the mobile devices to reduce the data size transfered over the Internet. The common mobile device has relatively limited computation capability and is not suitable for computationally-intensive applications such as FD. However, with help from certain System-on-Chips (SoCs), e.g. TMS320DM36x digital media system-on-chip (DMSoC) [35] from Texas Instruments, detecting faces on the mobile is significantly accelerated without consuming much CPU time and energy. In modern mobile operating systems, e.g., Android 4.0 Ice Cream Sandwich, the FD function has became a built-in API for the camera module [36], and with the help from the

aforementioned SoCs, the face detection can be done in real-time while previewing from the camera.

However, even with the help from such dedicated SoCs, mobile devices still have limited capabilities, not to mention those legacy mobile phones that have no such accelerator. For example, the mobile device we used in our mobile-cloud implementation, the LG Nexus 4 with Android 4.2 Jelly Bean, is only capable of detecting two faces in real-time. This use of face detection is mainly to support camera focusing. Also, the computational complexity of the FD is related to the image resolution. Most mainstream phones released after 2011 are capable of taking 720p video which implies a resolution of 1280x720. A high resolution image is desired if one wants to recognize people from the masses, which is very important during complex and demanding situations. Our work with the OpenCV library shows that detecting multiple faces in a high resolution image using the aforementioned Viola-Jones detection framework is very computation intensive and not a trivial task even for a modern powerful Intel x86 CPU (see Figure 4.3) . With support from Intel Threading Building Blocks (TBB) [37], the multi-thread CPU version of face detection takes about 1500ms to detect 17 faces in a single image (1280x720). This clearly shows that doing face detection is computationally prohibitive for mobile devices.

Based on the analysis above, we have determined that the mobile devices are not suitable for preprocessing therefore in our Cloud Vision system, the mobile devices will only capture the images and send them out for processing. For comparison, we also implement and evaluate the mobile-perform-preprocessing case even though maximum 2 detected faces per frame is hardly acceptable in real use cases.

4.2.2 Cloudlet

Since face detection operation is database-independent, it is separable from the subsequent steps and can be done by any device with proper computation power. Since our analysis above shows that the mobile devices are not capable of perform such preprocessing operation, it is necessary to add an intermediate node, i.e., a cloudlet to achieve the goal of real-time face detection.

The cloudlet, sitting on a LAN environment is more capable of accepting high speed data uploading than remote cloud servers. Also, the cloudlet can be much more powerful than the mobile in terms of computation. Based on our results (Figure 4.3), FD on the same image with an Nvidia Geforece [38] GTX 480 GPU (which was released in 2010 as the highest end GPU) takes 125 ms and on a GT 520 GPU takes 1451 ms. We observe these values to be proportional to the peak GFLOPS compute-power of the corresponding GPU. Thus, we can estimate the time that the FD operation will take 53 ms and 29 ms on other more recent GPUs, such as GTX 680 and GTX 690, respectively (see result marked with * in Figure 4.3).

As the semiconductor technology is marching towards the sub-20nm node, we can expect the peak GFLOPS for newer GPUs to increase, thereby reducing the FD time in each new generation. Since the communication delays cannot follow the same trend of rapid exponential reduction, we expect the acceleration provided by the cloudlet to improve with each new generation of GPUs. One other experiment we conducted is measuring the FD operation on a CPU-only computer. The result is far from acceptable with a 1500 ms response time, as reported in Figure 4.3. This shows that, the FD operation, can only be accelerated by a GPU and should not be performed on a CPU.

To further reduce the data size over the Internet, projection operation can also be included in the preprocessing. Given the fact that the savings on the data size

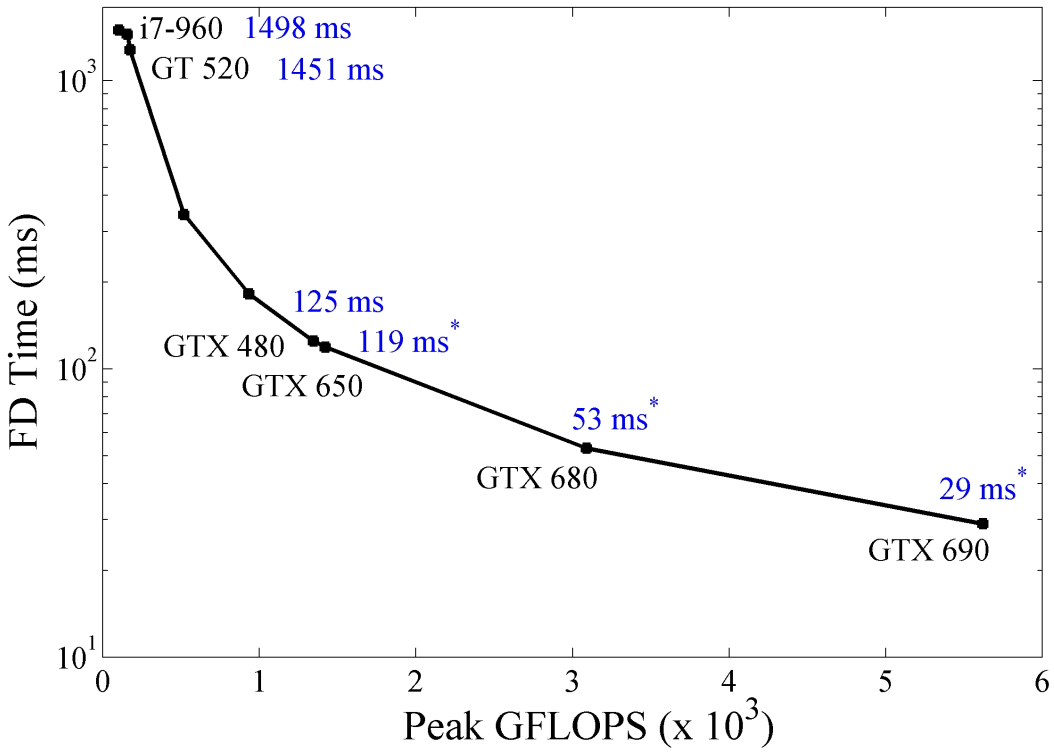

Figure 4.3: Real and estimated (marked with *) processing time of FD on images (1280x720) using different processing units.

from PJ is not as significant as doing FD, the cloudlet can use its intelligence to decide whether it should perform the projection based on various factors including: database size, network conditions, local processing power, server load, etc.

4.2.3 Cloud

The database typically resides in the cloud servers for easy update and management. Also the initialization of the database is very computation-intensive and time-consuming, which makes it only practical to be done off-line by the cloud server. Database search is typically done by the cloud servers since the database is on the server side. After the server receives the eigenvectors from the cloudlet, it will traverse its database to find the best match and send the result back to the mobile through the cloudlet.

In our prototype, the entire database resides in a single cloud server for simplicity. However, a huge database like the face database for global criminals is typically stored in several servers distributed around the world. In this case, the recognition result from the cloud servers is the local optima. Extra operations needs to be done on cloudlet or mobile devices to obtain the final global optima.

The cloud server will also handle part of the projection operations if the cloudlet decides not to perform PJ and send the raw faces to the cloud.

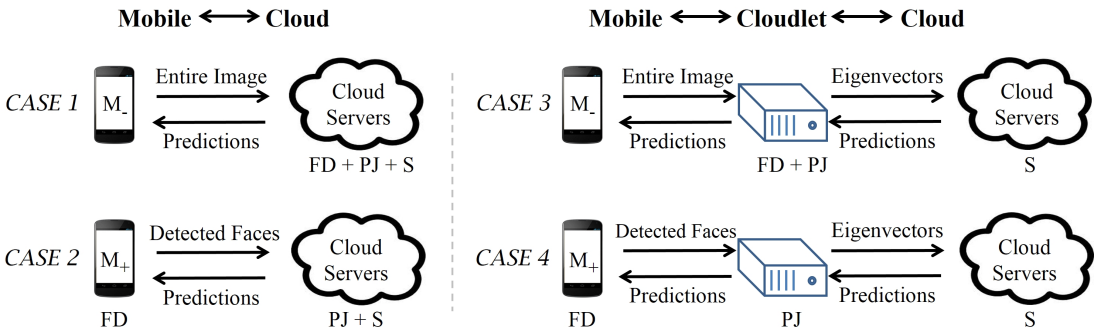

Figure 4.4: Four cases of mobile-cloud face recognition with/without a cloudlet. Detection (FD), projection (PJ) and searching (S), are distributed to different hardware components in each case. M_+ and M_- denote a mobile device with/without built-in face detection capability, respectively.

To summarize, our Cloud Vision system can be illustrated in Figure 4.4 as described above. The program on mobile device is developed in Java under Android platform while the program on cloudlet and cloud servers is developed in C using OpenCV library. The communication between the mobile, cloudlet and cloud is developed using the socket protocol.

5 Performance Evaluation

As described previously, some recent mobile devices are equipped with dedicated accelerator for face detection, which has significant effect on the overall face recognition application. To evaluate both cases, a mobile device with FD accelerator is denoted as M_+ and a mobile devices without such an accelerator is denoted as M_- throughout our evaluation.

5.1 Experimental Setup

To represent the M_+ case, we chose an LG Nexus 4 smartphone with Android 4.2 Jelly Bean as our mobile device, which supports a built-in real-time FD API, though it is only capable of detecting up to two faces per frame. The mobile device compresses the entire acquired frame (in the M_- case), or up to two faces (in the M_+ case) to JPEG and sends them to the cloudlet for processing. For M_+ devices, the Android API makes its best efforts to acquire, compress and send every frame, but certain incoming frames might be skipped if the hardware acceleratorion limit has been reached. In our experiments, the rate was around 7 fps at a resolution of 1280x720.

As analyzed in Chapter 4, the cloudlet is an intermediate node with high-speed connections to the mobile device (e.g., 802.11n), with a two-order-of-magnitude higher processing power than the mobile device. To meet this requirement, we

chose a Windows workstation with an Intel i7 CPU and a Nvidia GTX 480 GPU installed as our cloudlet. To isolate our system from the general public network traffic, we placed a dedicated wireless router between the mobile and the cloudlet. The cloudlet receives either raw images or images that contain only faces from the mobile, performs FD (in the case of receiving raw images) and PJ and sends the eigenvector to a cloud server. The connection between the cloudlet and the mobile is wireless LAN (WLAN).

We chose a Windows workstation that has almost the same computation power with the cloudlet as our cloud server. Though this is due to our hardware limitations, this configuration, however, only has a minimum affect on the experiment results. In our system, the cloud server is only responsible for S (in the case of mobile-cloudlet-cloud) or PJ and S (in the case of mobile-cloud). In theory, a more powerful cloud server could shorten both. But in the experiments, we found that even with our largest database (2000 images), the searching time is always less than 1ms and thus there would be no difference if we had used a more powerful cloud server. A shorter projection time is also not important for the comparison, because most of the performance gain comes from the pre-detection rather than pre-projection. If the cloud server is much more powerful than the cloudlet (and is not heavily loaded), the cloudlet can always choose to let the cloud server do the projection.

The cloud server receives 1) the eigenvector from the cloudlet and performs S, 2) raw images from the mobile and performs FD, PJ and S, or 3) faces from the mobile and perform PJ and S, and finally sends the result back. Although the connection between the cloudlet and the cloud server is LAN in our experiment, we added latencies to the response time on the cloud server to simulate real WAN conditions. The latency data were obtained through our experiments as described in our previously work [1]. Real latency data are obtained from experiments conducted on Planet-Lab where computers are physically located all over the

world. Since, in real life, mobile phones are always served by cloud servers located on the same continent, we chose three servers in North America: Albuquerque in New Mexico, Ottawa in Canada and Santa Barbara in California. The networking latency between us (Rochester, NY) and the three servers is shown in Figure 4.2.

All the hardware used in the experiments is listed in Table 5.1.

	CPU Freq	**#Cores/#Threads**	**GPU**	**Memory**
Mobile	1.5GHz	4/4	-	2GB
Cloudlet	2.9GHz	4/8	GTX480	12GB
Cloud	3.2GHz	4/8	Tesla C2075	24GB

Table 5.1: Specifications of hardware used in our experiments.

The database we used is the MUCT face database [39] which contains 273 distinct individuals with 10 to 15 images each. The initialization of the database is done off-line and therefore is not taken in account in the following evaluation.

5.2 Experimental Results

As illustrated in Figure 4.4, the baseline is when there is no cloudlet, the mobile sends the entire image or faces to the cloud. We are interested in the performance gain when we add a cloudlet to do the pre-processing. Figure 5.1 shows the average response time of the FR application under different image resolutions and different cloud servers for Case 1 - 4 illustrated in Figure 4.4. As we can see from the result of Case 1 and Case 3, with the help of the cloudlet, while a small portion of the response time is increased under low resolution compared with the system without the cloudlet, there is a significant performance gain (much lower response time) when the image resolution is high. This is because more redundant information is included in the original image as the its resolution increases. Doing pre-processing on the cloudlet can filter out that extra redundant

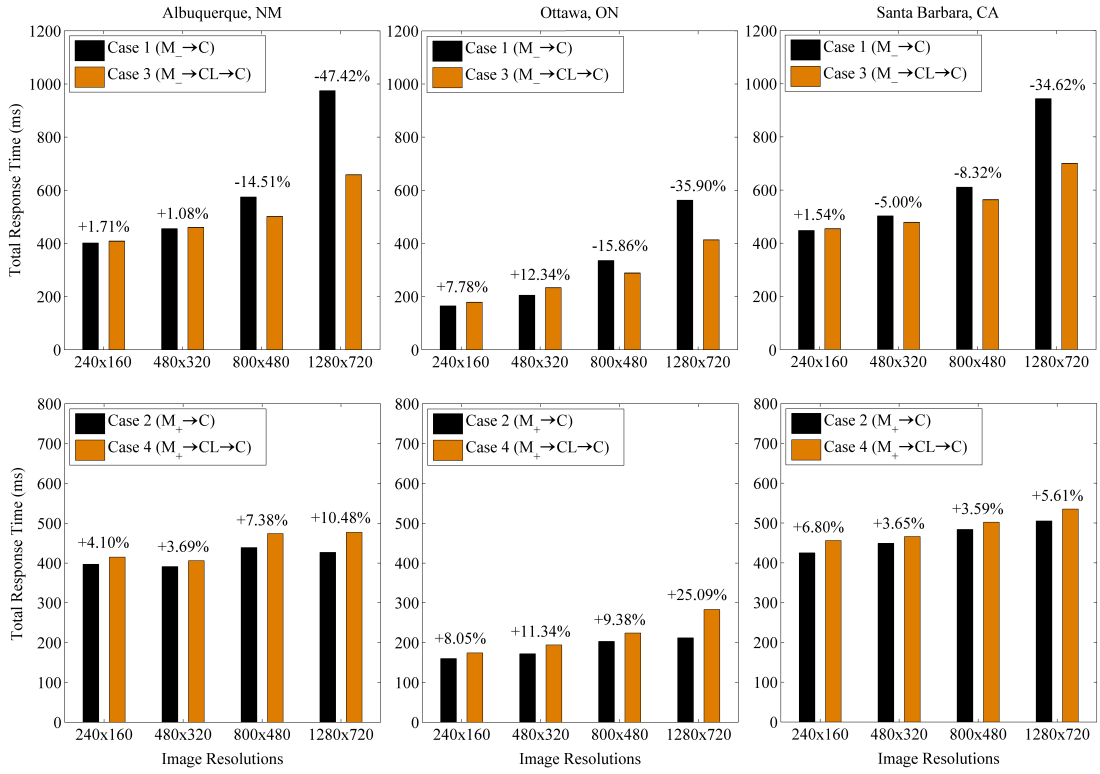

Figure 5.1: Response time of the FR application under different image resolutions and different cloud servers for Case 1 - 4 in Figure 4.4, here the size of the database is fixed to 50 images.

information before it gets transmitted on the Internet and therefore reduce the overall communication delay.

For the result of Case 2 and Case 4 shown in Figure 5.1, we do not see any benefits from the existence of the cloudlet. This is because the mobile has done the pre-detection and a large portion of the redundant information (the background) has been discarded before sending out. There is very little that the cloudlet can help with pre-processing. Additionally, the overhead (the time spent on decoding and encoding the packet, doing projection, etc.) added by the cloudlet even makes the overall performance worse. But as we mentioned before, doing face detection on the mobile cannot achieve very good results. A state-of-the-art smartphone with a dedicated accelerator (e.g., LG Nexus 4) can only detect two faces in each

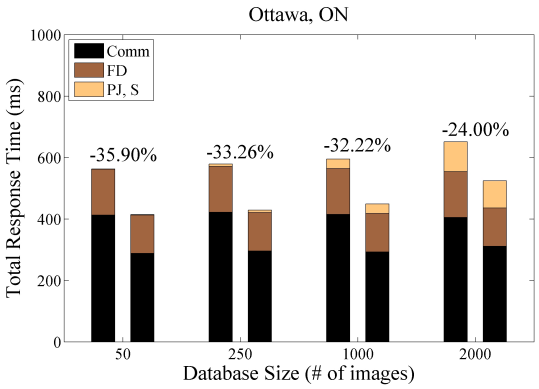

Figure 5.2: Response time of the FR application under different database sizes for Case 1 (the left bar) and Case 3 (the right bar) in Figure 4.4, here we fix the image resolution to 1280x720 and server location to Ottawa, ON.

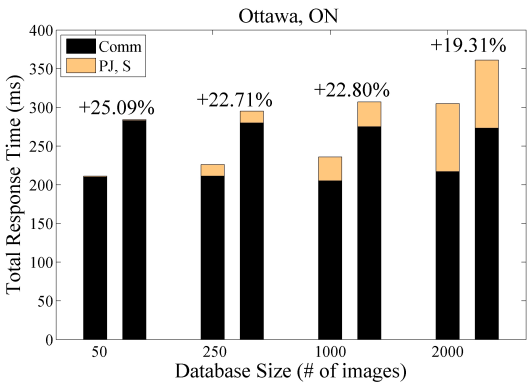

Figure 5.3: Response time of the FR application under different database sizes for Case 2 (the left bar) and Case 4 (the right bar) in Figure 4.4, here we fix the image resolution to 1280x720 and server location to Ottawa, ON.

frame. If one wants better detection results, a more powerful system must be used. A cloudlet can do the work without adding much communication delay.

A further study on the effect of the existence of a cloudlet is whether the relative performance changes when the database size increases. Figure 5.2 and Figure 5.3 shows the response time of the FR application as the database size increases. As we can see, the difference of the response time between systems with and without the cloudlet stays almost the same despite the changes in the size of the database. This is because the benefits of the cloudlet comes from pre-processing (especially from pre-detection, which is completely independent of the size of the face database).

6 Discussion and Future Work

Since the goal of this system is to solve the performance problem, we didn't take the energy consumption issues into account. Besides, compared with other systems like Kimberley, MAUI and CloneCloud, the Cloud Vision system removes the VM layer, which results in manual program partition and lack of manageability on the cloudlet. However, without the overhead from the synthesis and migration operation of the VM layer , Cloud Vision provides significantly higher performance, which fulfills the demanding requirements of the mission critical applications like real-time face recognition in the battlefield, where computation is extremely intensive and the response time is so critical that we cannot afford the overhead of VM-based system, and where we may have dedicated program and cloudlets, eliminating the manageability and programmability issues. One potential application area of such cloudlet-based systems is supercapacitor-based field systems with significant energy/power limitations [40] where the energy is of primary importance.

After a detailed analysis of the computational requirements of the face detection, we determined that, the FD operation can only benefit from the cloudlet with a high-powered GPU (e.g., 500 to 1000 peak GFLOPS) and even the most recent CPUs cannot provide an adequate acceleration. Since recent mobile devices have incorporated hardware acceleration for the FD operation, albeit only up to two faces per frame, the result obtained from M_+ shows the with-cloudlet-case

may have slightly poor performance than the without-cloudlet case. However, in the aforementioned battlefield scenario, the constrain for M_+ that, a maximumly 2 faces detection can be detected per frame, is not acceptable. To fulfill the critical performance requirements, the FD operation has to be done on the cloudlet.

Since the huge performance gap between a handheld device restricted by size and battery and powerful cloudlet and cloud servers will remain existing in the near future, the benefits provided by the Cloud Vision system will remain. To maintain significant performance gap between mobile devices and the cloudlet, it is necessary for the cloudlet to have significantly high powered GPUs. Since the FD operation time grows minimally when there are multiple faces in a picture frame (e.g., 50), we project that, a cloudlet based approach can have significant benefits when face recognition is being performed in scenes where there may be hundreds of faces and the cloudlet is equipped with a very high-powered GPU, such as the recent GTX 690 at 5600 peak GFLOPS. On the contrary, we found the CPU type to be nearly irrelevant, since the CPU in this application is merely executing the TCP/IP data shuttling threads as well as data movement, rather than any compute-intensive operation.

Although all of the attention was given to *performance* in this thesis, straightforward reformulation of the proposed algorithms to favor energy savings are also a viable future research direction. Alternatively, although pure software-based approaches have been presented in this thesis, hardware based accelerators to achieve algorithmic speed-up [41, 42] also deserve investigation.

Our future work plans include conducting experiments with commercial cloud servers like AWS, as well as a study of the other two aspects where the cloudlet can be helpful in terms of overall performance: 1) reducing response time by caching a portion of the database in the cloudlet, and 2) applying a scheduling algorithm on the cloudlet to optimize the service for multiple mobile phones. The energy consumption statistics will also be studied in our future work.

7 Conclusion

In this paper, we determine three approaches for accelerating the execution of the face recognition application by utilizing an intermediate device called a *cloudlet*. We study in detail one of these approaches, using the cloudlet to perform *pre-processing*, and quantify the maximum attainable response time acceleration. We presented a mobile-cloudlet-cloud architecture to perform real-time face recognition by executing this application in three distinct steps: face detection (FD), projection (P) and searching (S). We observed that, due to their separability, these three steps can be executed in different hardware components: mobile (M), cloudlet (CL), and cloud (C). While M and CL components are connected through a single-hop communication link, allowing large chunks of data (e.g., 500 KB) in a small amount of time (e.g., 10 ms.), the link between CL and C is much slower due to the multi-hop Internet connection. We provided a detailed study of utilizing a cloudlet to accelerate one of the three aforementioned operations and determined that, the FD operation is the one that can benefit most from a hardware-based acceleration. Our results show up to 47% acceleration when appropriate cloudlet hardware is used.

Bibliography

[1] Z. Dou, "Benefits of Utilizing an Edge Server (Cloudlet) in the MOCHA Architecture," Master's thesis, University of Rochester. Department of Electrical and Computer Engineering, Rochester, NY, 2013.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14 –23, oct.-dec. 2009.

[3] MOCHA, "Mocha research project." `http://themochaproject.com/index.html`.

[4] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision: Real-Time face recognition using a Mobile-Cloudlet-Cloud acceleration architecture," in *Proceedings of the 17th IEEE Symposium on Computers and Communications (IEEE ISCC 2012)*, (Cappadocia, Turkey), pp. 59–66, Jul 2012.

[5] T. Soyata, R. Muraleedharan, S. Ames, J. H. Langdon, C. Funai, M. Kwon, and W. B. Heinzelman, "Combat: mobile cloud-based compute/communications infrastructure for battlefield applications," in *Proceedings of SPIE*, vol. 8403, pp. 84030K–84030K, May 2012.

[6] "OpenCV Library." `http://opencv.org/`.

[7] PlanetLab, "Planetlab." `http://www.planet-lab.org/`.

[8] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2013. ¡ce:title¿Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures¡/ce:title¿.

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

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

[11] E. Chen, S. Ogata, and K. Horikawa, "Offloading android applications to the cloud without customizing android," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pp. 788 –793, march 2012.

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

[13] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating mobile cloud computing: A survey," in *Communication Infrastructures for Cloud Computing* (H. T. Mouftah and B. Kantarci, eds.), Hershey, PA, USA: IGI Global, 2013.

[14] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik, "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, (New York, NY, USA), pp. 23–28, 2012.

[15] D. T. Hoang, D. Niyato, and P. Wang, "Optimal admission control policy for mobile cloud computing hotspot with cloudlet," in *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pp. 3145–3149, 2012.

[16] S. S. Laurent, E. Dumbill, and J. Johnston, *Programming Web Services with XML-RPC*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[17] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[18] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st ed., 2009.

[19] E. E. Marinelli, "Hyrax: Cloud Computing on Mobile Devices using MapReduce," Sept. 2009.

[20] "Native Boinc for Android." `http://nativeboinc.org/`.

[21] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004.

[22] J. R. Eastlack, "Extending Volunteer Computing to Mobile Devices," Master's thesis, New Mexico State University, Carlsbad, NM, 2011.

[23] H. Ba, W. Heinzelman, C.-A. Janssen, and J. Shi, "Mobile computing - a green computing resource," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pp. 4451–4456, 2013.

[24] Wikipedia, "Augmented reality." `https://en.wikipedia.org/wiki/Augmented_reality`.

[25] S. Srinivasan, Z. Fang, R. Iyer, S. Zhang, M. Espig, D. Newell, D. Cermak, Y. Wu, I. Kozintsev, and H. Haussecker, "Performance characterization and optimization of mobile augmented reality on handheld platforms," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 128–137, 2009.

[26] T. Soyata, *Incorporating Circuit Level Information into the Retiming Process*. PhD thesis, University of Rochester. Department of Electrical and Computer Engineering, Rochester, NY, 1999. http://www.ece.rochester.edu/users/friedman/papers/Tolga_Soyata_PhD.pdf.

[27] T. Soyata and E. G. Friedman, "Incorporating Interconnect, Register and Clock Distribution Delays into the Retiming Process," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, vol. CAD-16, pp. 105–120, Jan 1997.

[28] T. Soyata, E. G. Friedman, and J. H. Mulligan, "Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 1748–1751, May 1995.

[29] T. Soyata and E. G. Friedman, "Retiming with non-zero clock skew, variable register and interconnect delay," in *Proceedings of the IEEE Conference on Computer-Aided Design*, pp. 234–241, Nov 1994.

[30] T. Soyata and E. G. Friedman, "Synchronous performance and reliability improvements in pipelined asics," in *Proceedings of the IEEE ASIC Conference*, pp. 383–390, Sep 1994.

[31] T. Soyata, E. G. Friedman, and J. H. Mulligan, "Integration of clock skew and register delays into a retiming algorithm," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 1483–1486, May 1993.

[32] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, (New York, NY, USA), pp. 13–18, ACM, 2012.

[33] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–511–I–518 vol.1, IEEE, 2001.

[34] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[35] "TMS320DM36x Digital Media System-on-Chip (DMSoC) Face Detection User's Guide." `http://www.ti.com/lit/ug/sprugg8a/sprugg8a.pdf`.

[36] "Android API Reference." `http://developer.android.com/reference/`.

[37] "Intel Threading Building Blocks (Intel TBB)." `http://threadingbuildingblocks.org/`.

[38] "GeForce Graphics Processors (GPUs) — NVIDIA GeForce — NVIDIA." `http://www.nvidia.com/object/geforce_family.html`.

[39] S. Milborrow, J. Morkel, and F. Nicolls, "The MUCT Landmarked Face Database," *Pattern Recognition Association of South Africa*, 2010.

[40] A. Fahad, T. Soyata, T. Wang, G. Sharma, W. Heinzelman, and K. Shen, "SOLARCAP: super capacitor buffering of solar energy for self-sustainable

field systems," in *Proceedings of the 25th IEEE International System-on-Chip Conference*, (Niagara Falls, NY), pp. 236–241, Sep 2012.

[41] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing," in *Proceedings of the International Symposium on Computer Architecture*, vol. 38, (Saint-Malo, France), pp. 371–382, Jun 2010.

[42] T. Soyata and J. Liobe, "pbCAM: probabilistically-banked content addressable memory," in *Proceedings of the 25th IEEE International System-on-Chip Conference*, (Niagara Falls, NY), pp. 27–32, Sep 2012.