A FLEXIBLE FINE-GRAINED ADAPTIVE FRAMEWORK FOR PARALLEL MOBILE HYBRID CLOUD APPLICATIONS

BY

SAEIDREZA SHIFTEH FAR

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Gul Agha, Chair
Professor Tarek F. Abdelzaher
Professor Nitin Vaidya
Professor Carlos A. Varela, Rensselaer Polytechnic Institute

ABSTRACT

Mobile devices have become ubiquitous and provide ever richer content and functionality. At the same time, applications are also becoming more complex and require ever increasing amount of computational power and energy. With cloud computing providing unlimited elastic on-demand resources, supporting mobile devices with cloud allows overcoming limitations of mobile devices. This is generally known as Mobile Cloud Computing (MCC) *and can be achieved through code offloading that selects computationally or data intensive parts of an application, outsources them to more-resourceful spaces and brings back the final results. While* code offloading *has been widely studied in the past within the context of distributed systems and grid computing, applying it to current mobile applications requires significant amount of manual changes to existing application codes. An alternative is to outsource the entire application process or the whole virtual machine in which the application is running. This solution assumes running the same code on a more-resourceful system is more efficient, but it is coarse-grained and requires significant amount of data to be transferred. Furthermore, requirements and expectations from mobile applications vary considerably by different users using wide range of mobile devices in various environmental conditions. This diversity in requirements and expectations creates wide range of target offloading goals, ranging from maximizing application performance to minimizing mobile energy consumption. The increased dynamicity and complexity of mobile cloud applications requires open systems that interact with the environment while addressing application-specific constraints, user expectations and hardware limitations.*

Our goal is to facilitate mobile cloud application development by masking all the complexity of mobile-to-cloud code offloading without requiring application developers to rewrite their code or perform additional manual work. Our focus is on separating the application logic, to be developed by programmers, from the application component configuration and distribution, to be adjusted transparently and dynamically at run-time. Our framework is fine-grained, supporting mobile application configuration and distribution at the granularity of individual components; it is flexible, allowing organizations, application developers, or end-users easily adjust target offloading goal or define policy-driven restrictions on offload-

ing budget, execution quality, or privacy and move around of components without modifying the existing application codes; and it is adaptive, addressing the dynamicity in run-time conditions and end-user contexts. It further supports component distribution in a hybrid cloud environment consisting of multiple public and private cloud spaces. Finally, it provides a new code offloading model that supports fully parallel program execution, where application components located at mobile device and multiple cloud spaces are executed independently but concurrently.

The proposed solution can be divided into three main parts: First, a light-weight monitoring system, called Monitor*, to capture dynamic environmental parameters and end-user context, profile application resource usage and communications, as well as monitoring availability and performance of cloud resources. Profiling energy consumption per specific application components is primary of importance and requires design and development of a fine-grained automatic energy consumption model, as most mobile devices do not provide any tool for direct measurement of consumed energy and different applications with arbitrary number of components might be running at any time. Second, we design and implement two independent performance-based and energy-based models to enable transparent automatic configuration and distribution of application code and data components that address specific organization, application, and end-user requirements. These models leverage dynamic information from the* Monitor *on run-time parameters, energy and resource usage of different components, and application characteristics to optimize application performance or mobile energy consumption with respect to a predefined policy. Finally, we design and develop a proof-of-concept framework called* IMCM*, Illinois Mobile Cloud Management, that embodies the described components to enable fine-grained adaptive application component configuration and distribution, while providing flexibility in terms of adjusting desired target optimization goal or defining additional policy-driven constraints on offloading budget, quality of service per resource, and privacy. Evaluations are carried out using a suite of benchmark applications, including computationally-intensive, I/O-intensive, communication-intensive and combined multi-purpose applications. Compared to sequential execution on a mobile device, these empirical benchmarks using* IMCM *framework result in speedups or energy-savings factor of over 50 times.*

ACKNOWLEDGMENTS

I would like to express deepest gratitude to my advisor Dr. Gul Agha for his full support, expert guidance, understanding and encouragement thorughout my study and research. Without his incredible patience and timely wisdom and counsel, my thesis work would have been a frustrating and overwhelming pursuit. His advice on both research as well as on my career have been priceless. In addition, I express my appreciation to Dr. Tarek F. Abdelzaher, Dr. Nitin Vaidya, and Dr. Carlos A. Varela for having served on my committee. They made my defense an enjoyable moment and their thoughtful questions and comments were valued greatly.

A very special thanks goes out to Dr. Kirill Mechitov for helping me with my academic research during my graduate years at University of Illinois. I would also like to thank Mary Beth Kelley from academic office of computer science department for all her help, support and patience. In addition, a special thank you to Dr. Frank Boukamp who gave me the opportunity to join graduate program of University of Illinois in the first place.

Thanks also to my fellow graduate students at the Open Systems Laboratory (OSL) at Computer Science department of University of Illinois, especially Parya Moinzadeh, Minas Charalambides, Peter Dinges, and Donna Coleman.

Special thanks go to my numerous friends who helped me throughout this academic exploration, especially Dr. Mani Golparvar-Fard, Bita Vaezian, Hamid Tabdili, Dr. Arash Tajik, and Dr. Omid Fatemieh, who supported and encouraged me to strive towards my goal. I am very happy that, in many cases, my friendships with you have extended well beyond our shared time in Champaign.

I would also like to thank to my family. Words cannot express how grateful I am to my parents for all the sacrifices that they made on my behalf. Their love provided my inspiration and was my driving force. I owe them everything and wish I could show them just how much I love and appreciate them. At the end, I would like to express appreciation to my beloved wife and best friend, Sara Khosravinasr, without whose love, encouragement and support, I would not have finished this thesis.

Thank you all for being part of my journey. I will always be grateful for your support.

TABLE OF CONTENTS

LIST OF TABLES viii

LIST OF FIGURES ix

LIST OF ALGORITHMS xi

CHAPTER 1 INTRODUCTION 1
1.1 Thesis Statement 4
1.2 Contributions 5
1.3 Organization 7

CHAPTER 2 RELATED WORK 9
2.1 Code Offloading and Mobile-Cloud Computing 9
2.2 Mobile Device Energy Measurement and Power Model Generation 13
2.3 Privacy and Access Control in Cloud Computing Environment 16

CHAPTER 3 ARCHITECTURE OVERVIEW 21
3.1 System Representation 22
3.1.1 Cloud Model 22
3.1.2 Mobile-cloud Application Model 23
3.1.3 Mobile Hybrid Cloud Application Entities 24
3.2 IMCM: Illinois Mobile Cloud Management Middleware Framework 25
3.3 Evaluation 27
3.3.1 Experimental Setup 27
3.3.2 Program Corpus 28
3.3.3 Implementation 30
3.3.4 Effectiveness of IMCM in automatic detection of application run-time parameters and offloading appropriate components 31
3.3.5 Performance overhead of IMCM 32

CHAPTER 4 A DECISION-MAKING MODEL FOR PERFORMANCE-BASED CODE OFFLOADING OF FULLY-PARALLEL MOBILE HYBRID CLOUD APPLICATIONS 34
4.1 Offloading Decision for Sequential Applications to Single Remote Server . . . 34
4.1.1 Code Offloading Decision Ignoring the Offloading Process Overhead . 34

4.1.2 Code Offloading Decision Including Offloading Process Overhead . . . 36
4.1.3 Code Offloading Decision for Large Applications . . . 37
4.2 Code offloading and Parallelism . . . 39
4.3 Performance-based Offloading Decision Model for Parallel Applications to Hybrid Cloud Environment . . . 42
4.4 Experimental Results . . . 46
4.4.1 Effect of run-time parameters on mobile-cloud offloading decision . . . 46
4.4.2 Effect of application type on mobile-cloud offloading decision . . . 47
4.4.3 Effect of problem size (amount of work) on mobile-cloud offloading . . . 48
4.4.4 Comparison of Sequential Local Application Execution versus Parallel Local and Remote Execution . . . 49

CHAPTER 5 A DECISION-MAKING MODEL FOR ENERGY-BASED CODE OFFLOADING . . . 55
5.1 Offloading Decision for Sequential Applications to Single Remote Server . . . 55
5.1.1 Energy-based Offloading Decision Model for Sequential Applications Ignoring Offloading Process Overhead . . . 55
5.2 Energy-based Offloading Decision Model for Sequential Applications Including Offloading Process Overhead . . . 56
5.3 Energy-based Offloading Decision Model for Parallel Applications to Hybrid Cloud Environment . . . 57
5.4 Experimental Results . . . 61
5.4.1 Effect of application type on mobile-cloud offloading decision . . . 61
5.4.2 Effect of problem size (amount of work) on energy-based mobile-cloud offloading . . . 61
5.4.3 Effect of application parallelism degree on energy-based mobile-cloud offloading . . . 61

CHAPTER 6 MONITORING APPLICATION COMPONENT ENERGY CONSUMPTION . . . 65
6.1 Energy Consumption of Applications . . . 65
6.1.1 Development of the Energy Estimation Model . . . 66
6.1.2 Calibration of the Energy Estimation Model . . . 68
6.1.3 Experimental Result . . . 71
6.2 Fine-grained Energy Consumption of Application Components . . . 72

CHAPTER 7 FLEXIBLE POLICY-DRIVEN RESTRICTIONS FOR PRIVACY, BUDGET LIMITATION, AND EXECUTION QUALITY OF MOBILE HYBRID CLOUD APPLICATIONS . . . 75
7.1 Defining Privacy for Mobile Hybrid Cloud Applications . . . 75
7.1.1 Design of the Authorization System . . . 76
7.1.2 Mobile Hybrid Cloud Authorization Grammar . . . 78
7.1.3 Grouping, Selection, and Binding . . . 80
7.1.4 Policy Description . . . 81
7.1.5 Policy Evaluation . . . 82

7.1.6 Implementation of the Policy-based Authorization System 84
7.1.7 An Example 86
7.2 Defining Execution Quality for Different Components at Various Locations . 97
7.3 Defining Offloading Budget Limitations for Different Cloud Resources 100

CHAPTER 8 CONCLUSION AND FUTURE WORK 113
8.1 Summary 113
8.2 Limitations and Future Work 115

REFERENCES 117

LIST OF TABLES

2.1 A comparison of Mobile-Cloud systems. 12

3.1 Specifications of the used equipment for evaluation 28
3.2 Benchmark applications used to evaluate our framework. 29
3.3 Benchmark application main characteristics showing dominant behavior of the application. 29

4.1 Speedup resulting from offloading 35
4.2 Components of a large mobile application and their interactions with each other 38
4.3 Notations used in parallel offloading model 43
4.4 Average bandwidth of different mobile technologies 47

5.1 Energy saving ratio resulting from offloading 56

6.1 Power estimation model for different mobile hardware components 74

7.1 Definition of authorization-related entities 79
7.2 Definition of authorization entity grouping, selection, and binding 81
7.3 Definition of authorization rules and their enforcement ordering 82
7.4 Private cloud policy defined as part of the hard policy file 107
7.5 Public cloud policy defined as part of the hard policy file 108
7.6 End-user application policy defined as part of the hard policy file 109
7.7 End-user soft policy for users without specific privacy concerns 110
7.8 End-user soft policy for privacy-concerned users 111
7.9 End-user soft policy for extremely cautious users with utmost privacy concerns 112

LIST OF FIGURES

3.1 Actor model of computation 25
3.2 Cost of VM migration in a local network. 26
3.3 Overhead of SALSA actor creation. 31
3.4 Overhead of SALSA actor migration. 31
3.5 Speedup of local, remote, and elasticity manager 32
3.6 Elasticity manager overhead 33

4.1 A mobile Application represented as a graph before partitioning. 50
4.2 A mobile Application represented as a graph after partitioning. 51
4.3 Speedup for NQueen problem 51
4.4 Effect of Image Quality on Speedup for Face Detection algorithm 52
4.5 Offloading decision for optimizing application performance. 52
4.6 N-Queen problem with different amount of work. 53
4.7 Effect of amount of work on speedup 53
4.8 Speedup different problem size 54
4.9 Remote vs. local+remote execution 54

5.1 Energy usage with different amount of work 62
5.2 Energy saving ratio with different amount of work 62
5.3 Speedup for different amount of work 63
5.4 Energy usage for local and remote execution 63
5.5 Energy saving ratio for local and remote execution 64
5.6 Parallelism and speedup for local and remote execution 64

6.1 Trend of remaining battery charge and battery voltage changes over time . . 69
6.2 Trend of battery voltage and AndroidSalsa consumed Energy changes versus remaining battery charge 70
6.3 Required steps to calibrate the energy estimation model 71
6.4 Energy profiling per application results 72
6.5 Estimated and measured powers 73
6.6 Estimated and measured energy 73

7.1 Policy schema tree 86
7.2 Part of policy schema definition as XML schema design file (xsd) 87
7.3 Actor entity attributes tree 88

7.4 Actor entity attributes definition as XML schema design file (xsd) 89
7.5 Modules involved in image processing application 90
7.6 Modules involved in image processing application 91
7.7 Internal structure and organization of databases within the private and public cloud spaces 92
7.8 The structure of the private cloud space consisting of 2 ActorSystems 93
7.9 Structure of the public cloud space. 94
7.10 Application for users without specific privacy concerns 95
7.11 Application for privacy-concerned users 96
7.12 Application for extremely cautious users 97
7.13 Local and remote execution of Image processing with different fixed qualities. 98
7.14 Code offloading with fixed quality of service for all locations and limited offloading budget rate. 99
7.15 Code offloading with variable quality of service for different locations and limited offloading budget rate. 100
7.16 Code offloading with unlimited offloading budget. 101
7.17 Code offloading with a fixed total offloading budget. 102
7.18 Code offloading with a limited offloading budget rate. 103
7.19 Code offloading with unlimited offloading budget in a Hybrid Cloud. 104
7.20 Code offloading with fixed total offloading budget in a Hybrid Cloud. 105
7.21 Code offloading with fixed offloading budget rate in a Hybrid Cloud. 106

LIST OF ALGORITHMS

Pseudo-code for rule evaluation algorithm 85

CHAPTER 1

INTRODUCTION

Mobile devices are ubiquitous these days, but they are still constrained by their limited resources. Compared to laptops and desktops, mobile devices typically have slower hardware, more restricted network access, and limited access to energy. These limitations have created an increasing gap between the demand for more complex applications and the availability of required hardware resources[1]. Cloud computing has the potential to provide an efficient solution to overcome mobile device constraints and to address the ever-increasing complexity of the modern applications. Not surprisingly, it is the new computing paradigm, serving as the center of data storage and processing for many enterprises. Cloud computing provides elastic on-demand access to virtually unlimited resources at an affordable price. Having access to elastic resources allows weaker devices run more-demanding applications by outsourcing storage or computation needs to cloud spaces. To achieve this, certain parts of mobile application have to be selected, sent to remote cloud spaces, executed, and the results brought back to the mobile device. This process is known as *code offloading* and has been widely studied in the past within the context of distributed systems and grid computing (e.g. [2, 3, 4]). Current practical solutions for providing the code offloading capability for mobile cloud applications rely on two main approaches:

1. *Hard-coding* the offloading decisions as part of the developed program
2. Using full *process or virtual machine (VM) migration* to make an exact copy of the running application within the cloud space

Hard-coding offloading decisions has the advantage that the offloading may be fine-grained, well-tuned, and potentially self-adapting based on run-time parameters, but such hard-coding requires programmers to rewrite their existing mobile application code in an offloadable mobile-cloud compatible format. Thus, hard-coding places a significant burden on application developers and requires continuous maintenance, as mobile applications evolve over time. In most cases, hard-coding makes the already complex programs even more complex and requires significant structural changes for existing applications. On the other

hand, VM migration approach is based on the assumption that running the same code on a faster machine improves the overall application performance. VM or process migration has the advantage of not creating additional work for developers but is highly coarse-grained. However, virtual machines are large components and moving them around is very expensive even when performed within a local area network (LAN) [5]. These limitations highlight the need for a transparent *fine-grained* solution that minimizes the required manual changes to applications and prevents creation of additional work for programmers.

Mobile applications are used by a wide range of users with different expectations, on mobile devices with different hardware capabilities, and under diverse environmental conditions. In addition to adapting an initial target offloading goal during execution, additional policy-based constraints may be needed to enforce privacy and restrict migration of components, limit bandwidth or energy use, or improve quality of service. As a result, application component distribution between mobile device and cloud resources must be *flexible* to satisfy different required restrictions and expectations at different times, and *adaptive* to address dynamic run-time environmental changes. This requires open systems that interact with the environment while addressing application constraints, user expectations, and hardware limitations. Current mobile application designs have been following one of the following two popular architectures [6, 7, 8, 9, 10, 11]

1. *Offline model*, also known as fat-client, that results in native applications where all parts of the mobile application are installed and executed locally on the phone. Even if remote data is required, local application periodically gets those data from remote servers and stores it locally on the phone. A good example is a stand-alone mobile game that is installed locally on the phone and works without requiring any further external data or resources.

2. *Client-Server model*, also known as thin-client, that results in current web applications where the mobile device provides only the minimum interface to connect to a remote service. A good example is using mobile browser to connect to Facebook remote services.

The need for a flexible fine-grained adaptive solution requires a new mobile application model that lies between these two architectures, i.e. that enables components running on and migrating between mobile device and cloud spaces. Despite some theoretical support for opportunistic parallelism, existing code offloading solutions pause local mobile execution while waiting for offloaded code result leading to semi-sequential application execution [12, 13, 14]. With modern mobile devices benefit from fast powerful multi-core processors, new offloading solutions should also support fully-parallel application execution.

Security and privacy issues also raise when outsourcing part of an application to external machines. Cloud has traditionally been used as a closed trusted environment but public cloud resources are owned and operated by third party companies. Public clouds raise additional privacy concerns when used for storing confidential or sensitive parts of applications. A private cloud space owned by an individual enterprise allows keeping the enterprise's confidential or sensitive algorithms and user data within private secure in-house servers. Different users and organizations have different regulations and expectations in terms of the privacy of their data. As a result, cloud computing architectures have moved from a model that solely relies on public cloud spaces to a *hybrid cloud model* combining both public and private cloud spaces. Individual application users may even have further concerns about the privacy of their data. This requires a multi-level policy-driven approach in terms of defining additional required restrictions, governing allowable actions and migration of application components between mobile device and different cloud spaces.

The goal of this thesis is to bridge the gap between mobile application development, cloud computing and dynamic adaptive code offloading while satisfying both application and end-user requirements. Our main design objective is to help mobile cloud application programmers focus on developing their application logic without worrying about component distribution; Instead, component distribution would be performed transparently and dynamically at run-time. We propose a framework that masks all the complexity of mobile application code offloading to multiple cloud spaces. Our goal is to separate the application logic - to be developed in future by programmers - from the application component configuration and distribution - to be performed transparently and dynamically at run-time by a cloud management framework. Modeling mobile-cloud application as a composition of self-contained autonomous actor components, the thesis describes such a framework as fine-grained, supporting application configuration and distribution at the granularity of individual components; transparent, masking the underlying complexity of mobile-to-cloud code offloading; flexible, allowing end-user or application developers easily adjust target offloading goal or redefine constraints governing offloading budget, execution quality, or privacy and migration of components without modifying the existing application code; and adaptive, addressing the dynamicity in run-time conditions and end-user contexts. The framework further supports component distribution in a hybrid cloud environment consisting of multiple public and private cloud spaces. Finally, the framework provides a new code offloading model that supports fully parallel program execution, where application components located at mobile device and different cloud spaces are executed independently but concurrently.

We call our proposed framework Illinois Mobile Cloud Management framework, or *IMCM*. IMCM provides a light-weight monitoring system to capture dynamic environmental param-

eters and end-user context, profile application resource usage and communications, as well as monitoring availability and performance of cloud resources. Direct automatic profiling of energy consumption of a specific target application and its components is of primary importance. Note that mobile devices do not provide any tool for direct measurement of energy consumed by an application when different applications with arbitrary number of components may be running at any time. Thus, supporting automatic profiling of energy consumption requires design and development of a fine-grained automatic energy consumption model. IMCM also provides two independent performance-based and energy-based models to enable transparent automatic configuration and distribution of application code and data components that address specific end-user and application requirements. These models receive dynamic information from a monitoring module which measures run-time parameters, energy and resource usage of different components, and application characteristics in order to optimize application performance or mobile energy consumption. In contrast to existing solutions that only support offloading to a single remove server and serial monotonic application execution [12, 13, 14], IMCM supports a hybrid cloud environment, consisting of multiple public and private cloud spaces, in addition to fully parallel application execution. This results in significantly different component distribution plan based on a target offloading goal and requires the design and development of independent offloading models for each goal. IMCM provides required flexibility in terms of adjusting desired target optimization goal at run-time. Such goals may include multi-level policy-based constraints restricting privacy, offloading budget, and quality of service per resource.

IMCM performance is evaluated using a suite of benchmark applications including I/O-intensive, computationally-intensive, communication-intensive and combined multi-purpose applications. Evaluation benchmark applications are selected based on their main characteristics in a way that different extreme application behaviors are covered. In addition, a multi-behavior application is also considered that combines different possible behaviors and represents many existing ordinary applications.

1.1 Thesis Statement

This research focuses on the problem of code offloading from a mobile device to a hybrid cloud space in order to overcome the limitations of mobile devices. The ultimate goal is to improve end-user experience by transparently adjusting the mobile application based on user expectations and environmental parameters without creating additional work for application developers. Among the main axis of our design goals are *flexibility*, *fine-granularity*, *separa-*

tion of concerns, and more importantly, *adaptivity*. Our proposed solution can be divided into three parts: First, we design an adaptive fine-grained framework that allows automatic distribution of application code and data components, addressing specific end-user and application needs, while providing flexibility in terms of desired target optimization goal and required additional policy-based restrictions on privacy, offloading budget, and quality of service of resources. Second, we design and implement two independent performance-based and energy-based models to find optimum component distribution plan based on dynamic information from run-time parameters, energy and resource usage of different components, and application characteristics. Finally, we design and implement a light-weight monitoring system to capture dynamic environmental changes in end-user context, profile application resource usage, energy consumption and communications, as well as monitoring availability and performance of cloud resources. We summarize the thesis statement as follows:

Mobile cloud applications require restricted fine-grained adaptive and dynamic configuration and distribution of code and data components. Such adaptation can be achieved by online monitoring of user context, resources, and communications and solving constraints to update component configuration and distribution.

1.2 Contributions

To summarize, this research has the following main contributions:

- We have designed a flexible fine-grained adaptive framework called IMCM, that uses on-line monitoring, offloading decision-making models, and policy-driven constraints, to enable dynamic application configuration and adaptive component distribution. In our design, we follow principle of *separation of concerns* to separate development of application logic, to be developed by future programmers, from application component distribution plan and use the *Actor model of computation* to present code and data components and their interactions. *Actors* are concurrent objects interacting via asynchronous message passing. Our proposed framework provides a systematic method to support dynamic application component configuration and distribution for mobile-cloud applications based on run-time parameters and individual application requirements. Specifically, we design an approach using specification of application-defined requirements and user-defined expectations. Application target goals are significantly different and range from maximizing application performance to minimizing mobile energy consumption. We model each mobile application according to its dynamic target

goal and formulate the problem into an optimization problem. Every possible configuration of components that satisfy the requirements will be evaluated and the best distribution plan that optimizes the dynamic application target goal is selected and enforced. (Chapter 3).

- We have developed a component offloading decision-making model to maximize application performance for fully-parallel mobile hybrid cloud applications. The model is fine-grained and minimizes total execution time consisting of the time required to execute the application code in addition to the time required to communicate and exchange data. Fully parallel execution includes concurrent application execution on multiple remote cloud locations in addition to simultaneous local mobile and remote cloud execution. This requires combining the execution time of different components at various locations with their communication time in order to find the best distribution plan minimizing both. (Chapter 4).

- We have developed a component offloading decision-making model to minimize local mobile energy consumption for fully-parallel mobile hybrid cloud applications. While offloading to a single remote server and serial monotonic application execution results in similar offloading plan for both target goals of optimizing for application performance and mobile energy consumption, supporting hybrid cloud environment with multiple public and private cloud spaces in addition to fully parallel application execution results in significantly different component distribution plan for different offloading goals. This requires design and development of a very different offloading model. While performance-based decision-making model emphasizes more on details of communication between different components, energy-based decision-making model relies more on the effect of offloading on communications between mobile device and different cloud spaces. (Chapter 5).

- We have designed and implemented a light-weight, on-line monitoring system to profile run-time parameters, end-user context, application resource usage and communications, as well as availability and performance of mobile and cloud resources. We will then use these profiled data to predict how application components will work in future and how migration of different application components to different cloud resources will affect mobile energy consumption or application performance. One of the main challenges of monitoring module is to profile energy consumption of a specific application and for all its components, as most mobile devices do not provide any tool for direct measurement of energy consumption and different applications with arbitrary number

of components might be running at any time. In order to do this, we developed an automatic fine-grained energy consumption model to estimate consumed energy per applications and per application components. (Chapter 6).

- We have developed the required grammar and enforcement mechanism to allow enterprises, developers or end-users to easily adjust target offloading goal at run-time in addition to define required policy-based restrictions in terms of privacy, component migration, communications, resource accesses, offloading budget, and quality of service of resources. Privacy of data is a challenging issue especially when parts of the application is outsourced. Many companies are using a hybrid cloud model in order to keep the confidential or sensitive algorithms or user data within private secure in-house servers. Our provided grammar allows enterprises, application developers, and end users to define their required privacy authorization rules and adjust them during execution. A light-weight action control and policy management system is designed and implemented to interpret the defined policy rules and enforce them at run-time at different levels. The grammar it easily extendable to include many other required restriction. (Chapter 7).

1.3 Organization

The remaining of this thesis is organized as below:

In Chapter 2 we present a review of related work. Specifically, we overview previous work in the areas of *code offloading*, *mobile cloud computing*, *mobile energy model generation and energy consumption profiling*, in addition to *privacy and access control in cloud computing*.

In Chapter 3, we introduce our high level architecture and different used components. We leverage the *actor model of computation* to represent application code or data components and their interactions.

In Chapter 4, we review design and development of a performance-aware fine-grained decision-making model for application component distribution between mobile device and a hybrid cloud consisting of multiple private and public cloud spaces for a fully parallel application execution. Similarly, Chapter 5, discusses details of an energy-aware mobile hybrid cloud offloading decision-making model.

Chapter 6 discusses details of the light-weight on-line *monitoring* system. The goal of the monitoring system is to provide the rest of the system with necessary information for an efficient configuration in response to run-time, application and hardware changes. The

Monitor has three main goals. First, it should profile run-time parameters and end-user context. Second, it should profile application resource usage in addition to individual application's components resource usage and communications. Third, it should profile energy consumption per application component for which it requires an automatic energy estimation model.

In Chapter 7, we discuss the flexibility of the system in defining policy-driven constraints restricting privacy, offloading budget, and quality of service per resource. We, then, conclude the thesis in Chapter 8 and discuss future directions.

CHAPTER 2

RELATED WORK

Mobile applications are typically constrained by mobile hardware limitations. Code offloading allows outsourcing part of an application to more resourceful systems. Over time, code offloading techniques have evolved and have become an efficient solution to overcome mobile device limitations. However, there are several issues and concerns when offloading part of an application to remote servers. These issues range from heterogeneity in hardware platforms and varying environmental conditions to different user expectations and privacy issues resulting from running code on third party machines. Below, we review existing systems and solutions that aim to tackle these problems.

2.1 Code Offloading and Mobile-Cloud Computing

Code offloading refers to sending all or part of a computation to a more resourceful machine. This is not a new idea and the popular client-server model has been doing this for a long time where a thin client always sends computation to a server. It has also been used within grid computing where processes were migrated within the same computing environment for the purpose of load balancing between different machines [1, 15]. Prior to 2000, due to limitations of available technologies, researches were mostly focused on discussing the feasibility of offloading [16, 17, 1]. Later, between 2001 and 2007, with the appearance of PDA and other handheld devices, availability of reliable virtual machines and faster wireless network access, the focus of most study moved toward implementing working prototypes and developing algorithms for making offloading decisions [18, 19, 20, 1]. Modern offloading era started in 2007 where improved virtualization techniques became available, network bandwidths significantly improved and cheap cloud computing infrastructure appeared [21, 22, 23, 1]. During this time, two main types of system architectures became popular [24]: client-server or service oriented architecture and virtualization approach. Remote Procedure Call (RPC) and Remote Method Invocation (RMI) are examples of popular protocols that enabled inter-process communication between different machines in the service-oriented architecture. However, it

was virtualization approach that created modern offloading movement by allowing cloud vendors to run arbitrary applications from different customers on virtual machines instead of service providers managing programs running on servers [25].

In recent years, with the popularity of mobile devices and the availability of relatively cheap public cloud resources, code offloading has been extended to mobile devices and Mobile-Cloud Computing (MCC) was introduced to overcome mobile limitations by offloading part of applications to elastic cloud resources in a dynamic on-demand fashion [26]. For many years, most of the systems benefiting from code offloading used one of the following two approaches: Rely on the programmers to manually partition the program and specify how to offload parts of an application to remote servers [27] or to us full virtual machine migration [28, 29] in which entire process or entire OS is migrated to cloud space assuming that executing the same code on a faster machine improves the performance. Overall picture of modern elastic mobile-cloud applications research is presented by Zhang et al [30]. Zhang et al [30] suggest that an elastic application will consists of one or more weblets, each of which can be launched on a mobile device or a cloud space. Weblets can be migrated between mobile device and cloud space according to dynamic changes of the computing environment or user preferences on the device. However, their work discusses the high level idea and the requirement for such a system rather than provide a specific design or implementation. Moreover, the work primarily focuses on security concerns within such a system.

MAUI (from Microsoft research) [12] is one of the first working implementation of mobile code offloading that enables fine-grained energy-aware offloading of mobile code to cloud resources. MAUI uses a combination of virtual machine migration (VMM) and automatic code partitioning to decide at run-time what single method should be remotely executed in order to minimize energy consumption on mobile device. However, every time a single method is being offloaded, the program halts, the state is transferred to a remote machine, executed there and the result and new state is brought back to the mobile device, merged with the local program state before the execution of the program resumes. This results in completely sequential application execution, requires manual annotation of methods by programmer, and offline static analysis of the source code before execution. In order to overcome some of the limitations of MAUI such as the requirement for manual annotation of offloadable methods, CloneCloud [13] was introduced in 2011 by Intel Berkeley Labs: CloneCloud avoids manual annotation and enables unmodified mobile applications to offload part of their execution from mobile device into device clones running in a cloud space. CloneCloud automatically marks offloadable methods by static analysis of the application byte-code but dynamically decides at run-time about the optimal offloading plan using profiled data. Multiple methods may be offloaded at the same time. A significant limitation of CloneCloud is that it requires

an exact clone of the mobile device in the cloud and uses virtual machine migration (VMM) to transfer the memory image, CPU state, storage contents and network connections between the mobile device and the cloud clone. While theoretically CloneCloud provides opportunistic concurrency, in practice it results in an almost serial execution as the phone execution halt whenever a method tries to access migrated state or make a call to migrated functions. Moreover, the application partitioner is static and needs to pre-process the application code in an offline mode in order to determine offloadable parts of the code. CloneCloud also only considers limited input/environmental conditions in the offline-preprocessing and needs to be bootstrapped for every new application built. ThinkAir [14] supports method-level computation offloading and is designed to address MAUI's lack of scalability by providing on-demand virtual machine creation and to remove CloneCloud restrictions on applications, inputs, and environmental conditions by adopting an online method-level offloading that allows on-demand resource allocation. The main focus of ThinkCloud is on dynamic creation, resume, and destroy of virtual machines as needed. However, a single mobile device only knows and interact with one of these VMs; Other VMs are masked and only available to the primary VM. Also, an application needs to be modified and programmers have to manually annotate their offloadable methods. Moreover, VMs on the cloud will be used by different mobile application users and there is no isolation or security protection between offloadded code from different users. Similar to CloneCloud, ThinkAir also relies on existing opportunistic parallelism inside the mobile application and blocks mobile execution if there is any call made to the offloaded methods. In fact, ThinkAir is more focused on VM load-balancing rather than mobile-cloud application offloading. Load-balancing of VMs have been studied by many researchers [31, 32, 33, 34, 35, 36] with the primary focus on improving physical hardware resource utilization by moving or assigning VMs to physical machines with lower resource utilization and balancing load between different physical machines. Load balancing VMs results in better overall utilization of physical machines and is essential for cloud management but it does not improve the performance of a specific mobile application. Since VM migration is typically coarse-grained, Imai et al [5] combines VM migration with application-level migration to reach fine-grained load balancing. Their work is similar to ours as it is also based on component-based application development using actor programming paradigm and focuses on moving individual actors between different spaces. However, Imai et al [5] are also focused on load-balancing of VMs over physical machines within a cloud space and do not consider the side-effect of that on individual applications. Moreover, they focus on cloud environments and do not consider specific requirements of mobile applications, dynamic context of mobile devices, and varying requirements of application users over time.

Orleans is another framework for building elastic cloud applications developed by Microsoft research [37]. Orleans exploits notions such as immutability to enable state replication. Orleans is based on distributed actor-like components called grains. Grains are containers of the application state and can be persisted to durable storage. However, unlike actor model, multiple instances of a grain can concurrently run and modify the grain's state. Our framework differs from Orleans by maintaining a closer adherence to the actor model in that there is only one thread in the system at any given time that can change an object-actor's state. This is in contrast to Orleans which permits concurrent modification of multiple activation of a grain. This allows Orleans to have greater responsiveness and system throughput at the expense of a weaker consistency model [38]. In addition, Orleans supports only weak mobility: an actor may be moved from one machine to another but not while processing a request. It also does not provide fair scheduling and does not allow loading new code into an actor at run-time: actors are started with an initial code and will continue executing the same code. In fact, its current implementation only supports best-effort fair scheduling for multiple grains and multiple instances of each grain.

Table 2.1: A comparison of Mobile-Cloud systems. Goals of minimizing mobile energy consumption and maximizing application performance lead to same offloading plans when local mobile execution is paused while offloading code.

System	Year	Goal	Offload Decision	Partition Level	Parallel	Privacy	Manual work	No. Cloud spaces
MAUI	2010	Energy	Dynamic	Method	No	No	Yes	1
CloneCloud	2011	Energy= Performance	Static	Method	pseudo	No	No	1
ThinkAir	2012	Energy= Performance	Dynamic	Method	pseudo	No	Yes	1
Cloud OS (COS)	2012	Load Balancing	Dynamic	Actor	Yes	No	No	Many
Orleans	2014	State Recognition	Static	Grains	Yes	No	No	Many
IMCM	2014	Energy, Performance, Data usage, combination	Dynamic	Actor	Yes	Yes	No	Many

Table 2.1 compares our approach with that of others. In almost all of the mentioned mobile-cloud offloading systems, solving the offloading problem for maximizing energy saving on the mobile device or maximizing the application performance, or minimizing total application execution time results in almost same offloading plan. The reason is that mobile

device is assumed to be in idle state while offloaded code is being executed. As a result, the amount of energy consumed on the phone is proportional to the execution time and the energy-based and performance-based optimization goals results in the same offloading plan. This is an effect of running applications sequentially which is a basic assumption in design of MAUI [12]. However, even in systems that are trying to support opportunistic parallelism [13, 14], the execution of the application ends up being very close to sequential execution. The reason is that such systems pause on-mobile execution as soon as the on-mobile code makes a call to any of the offloaded methods or tries to access the state needed by the offloaded code. In fact, this is one of the main distinction between our research and previous work, as our application model is based on naturally concurrent actor model and supports full parallelism. The actor model provides completely autonomous code components that can be executed independently and concurrently. This significantly simplifies the system as the compile time method-level migration makes it too complex for most practical applications [39, 40]. Another distinction between our work and previous research is the fact that most of the previous systems are limited to offloading to a single remote resource. Although a few systems [14, 5] support dynamic creation and merging of VMs as needed, these systems are focused on improving underlying physical machine performance and not application component outsourcing to different cloud spaces because of privacy, cost, performance, or availability of resources. Finally, most of the previous studies make no distinction between a cloud space and a remote server. Our work is based on a general hybrid cloud model where multiple cloud resources with different parameters, costs, performance, and access levels are available.

2.2 Mobile Device Energy Measurement and Power Model Generation

Since most mobile devices do not provide any tool for direct measurement of consumed energy, energy consumption models using available parameters are required to estimate energy use in mobile devices. Older generation of power models work as black-boxes that require no knowledge of hardware components but rely on processor power models. These models are based on the assumption of linear relationships between processor power consumption and several hardware performance counters, e.g. instructions executed and TLB misses. PowerScope [41] works based on this method and assigns energy consumption to processes and procedures within a process without imposing large overhead. However, PowerScope only models power consumption by the CPU and its result shows only part of the real energy con-

sumed. Moreover, it requires programmers to use a set of specialized APIs to estimate power consumption. More recent models usually assume energy consumption of different hardware components independent and estimate total energy consumption based on measured usage of different components. As a result, these models are sensitive to the selected coefficients for estimating the contribution of each hardware component toward total energy consumption. An important step of generating an energy estimation model is to find appropriate coefficients for different hardware components. Methods for generating energy models for mobile devices can be categorized into manual and automated models. Most of prior work relies on manual methods involving external power meter equipment to calculate required coefficients for different hardware components energy consumption [42, 43]. Shye et al [42] proposed a utilization-based power model. Pathak et al [43] proposed a finite state machine (FSM) based power model using system call to overcome the limitation of utilization based power model. However, it is time consuming and hard to construct individual power models for each new mobile device using these manual approaches because of the need for external power metering equipment and significant amount of manual work.

Due to the difficulty of manually generating energy consumption models, automated methods for generating the models have become popular in recent years [44, 45, 46, 47, 48]. PowerProf [46] provides an unsupervised power profiling scheme for Nokia mobile devices that generates component power models based on a generic algorithm in order to automatically identify the power states of underlying hardware components. PowerProf enables online energy estimation, but the scheme is focused on power modeling rather than application energy metering. It measures power consumption for API calls issued in programming language and is limited in terms of application energy metering because the technique still strongly depends on the programmer's intention. Zhang et al [44] proposed using existing internal voltage sensor and a training software to generate the energy consumption model. They defined states for each hardware component and then executed a set of training programs for each defined state. The power consumed during the execution period is then sampled to generate the power model. Kim et al [48] overcame the limitation of PowerTutor [44] which considers only one CPU core and linear power consumption for the display. They proposed a more improved power estimation technique than PowerTutor, considering multi-core mobile devices and the nonlinear power consumption characteristics typical of displays and 3G modules. Our proposed model is similar to these models as they all depend on battery discharge curve and training programs to generate the power model. Processors and Screens consume most part of energy on a mobile device. However, there is significant difference between energy consumption of LCD screens, as considered in these models, and the OLED screen, as used in out experiments. LCD screens are lit up using a background light and

their energy consumption can be measured using the brightness of the background light. On the other hand, OLED and AMOLED screens do not have background light. Instead, each pixel is individually lit up. We used an average of brightness of all pixels on the AMOLED screen for estimating energy consumption resulting from screen.

Dong et al [45] suggested an accurate self-modeling mechanism for generating power estimation models without requiring any external equipment. Their method is able to achieve high accuracy rate (100 Hz) but it only targets the whole mobile system and not individual applications. Jung et al [47] proposed an autonomous power modeling tool for mobile devices, which overcomes the limitations of internal sensors. They generate their model according to the update rate of internal sensors but they rely on mobile current sensor for current values. Most mobile devices do not have an on-board current sensor limiting the application of their proposed solution. Lee et al [49] suggested a solution for generating power model without requiring detailed knowledge of the underlying hardware or a specific training program. Their method uses a regular user's usage pattern, which is already recorded on the phone, to identify data segments that contains the hardware status and power data for a single operating state. They then use these extracted data and regression analysis to build energy consumption estimate for each hardware component. Moinzadeh [50] also followed a similar approach and used application logs and regression to estimate energy consumption of individual components of a wireless sensor node. Compared to PowerTutor [44], these regression-based models [49, 50] use data from the user log file instead of a benchmark suite. However, their approach takes more time to generate a power model, as enough logged usage data must first be collected to perform regression analysis. We believe use of the calibration code is not a problem for most applications. Besides, generation of the power model needs to be performed only once for each mobile device. As a result, only one user needs to run the calibration code for each new mobile device and the generated energy model can be shared with other users.

Another view on automatic energy estimation models is based on their speed and frequency for reporting updates. Based on this view, energy estimation models can be classified into two categories: i) sampling-based methods ii) event-based methods. Sampling-based methods periodically collect the information of hardware activities and power modes from the Linux kernel and predict the system power consumption based on them. [42], [44], and [45] uses sampling-based methods. A common issue with all sampling-based techniques is the excessive performance and energy overhead due to frequent accessing the Linux kernel and reading large amount of data from the kernel even when there is not much changes in system parameters.

In contrast, event-driven methods collect system activities only when an event, which

affects the system power consumption, occurs. This approach significantly reduces the overhead of profiling and makes it more usable for high-frequency profiling of energy consumption. However, this method requires modifying the source code of Android or the Linux kernel that limits its application for most practical applications. Eprof [51] predicts the power consumption using this approach by modifying the kernel code and accounting system call events. Based on the FSM power model [43], Eprof is able to analyze the asynchronous energy state of an application, modeling the tail-state energy characteristics of hardware components with routine-level granularity. Energy metering is achieved using a post-processing mechanism using an explicit accounting policy. Eprof requires modification in the Android framework, to trace the API calls, in addition to modification to application source code. AppScope [52] uses the debugging tools for the Linux kernel to collect the required information in a non-disruptive way. Although they did not modify kernel code, they still insert breakpoints in arbitrary kernel functions in order to be notified of desired events. FEPMA [53] follows the same approach and provides highly accurate and nearly instantaneous estimation of power consumption by monitoring system events in the device driver layer of Linux kernel. However, they still rely on debugging tools to insert breakpoints at arbitrary functions of operating system kernel in order to be notified of the events. Some of the common drawbacks of event-driven approaches are the inaccurate event time-stamps, un-observable devices, and low time granularity of the power metering [53]. Since our application does not require very high frequency sampling of run-time parameters and in order to prevent any modification to kernel code, we decided to use a sampling-based method. This allows us to adjust the sampling rate as needed. Besides, the overhead of running our sampling-based power generation model is relatively low, as we only need power estimation every few seconds and based on the intervals of running the distribution optimizer.

2.3 Privacy and Access Control in Cloud Computing Environment

privacy and access control is a big concern when outsourcing parts of an application data or computation to machines owned and operated by third part companies [54, 55, 56]. Since our framework supports offloading data and computation from mobile device to a hybrid cloud model consisting of one or more cloud resources, we need a suitable cloud environment access control system to define and enforce the required restrictions and limitations in terms of offloading different parts of the mobile application to different cloud resources, interactions between different cloud resources, and interactions between different mobile-cloud application components when placed at different locations. We review existing cloud computing

access control systems in this section.

The widespread application of cloud platforms for managing large datasets has brought an increasing awareness of the security requirements of these applications [57, 54, 56]. Traditionally, cloud solutions are developed in a closed trusted environment without providing high level of security protection [54, 58]. Such a trusted environment can only be reached when the cloud is isolated from the outside word and used by a small group of authorized people within the company. However, with the cloud acting as the heart of many organizations and applications [30], the number of users accessing cloud services has increased dramatically. In addition to the internal users accessing the cloud from within the company, new external clients also interact with the cloud through the provided cloud-based applications. It is imperative to adopt a flexible and fine-grained authorization system to regulate accesses to different cloud resources. The required levels of accesses are different for internal users from different departments developing programs on top of the cloud and for external clients reaching the cloud through mobile-cloud applications running on their un-trusted personal devices. This highlights the need for a reliable access control system for cloud-based applications.

Sandhu et al. [57] defined authorization as limiting the actions that a legitimate user can perform within a system. It does not provide a comprehensive solution for all security concerns of a system and has to be coupled with authentication and audition in order to work. Authentication establishes the identity of users, authorization controls their accesses, and audition provides a posterior analysis of all the activities of the system. Samarati and di Vimercati [59] further decompose authorization into a security policy, defining what needs to be enforced, a security mechanism, enforcing the defined access control decisions, and a security model, providing a formal representation of the defined access control policy. Traditional authorization models of Discretionary Access Control (DAC) and Mandatory Access Control (MAC) (as explained in the Orange Book of the U.S. Department of Defense) are either too weak for effective control of information assets or are too rigid and often subverted in practice [60, 57]. Moreover, they are both static authorization systems while cloud applications require dynamic policies allowing addition and removing of subjects and resources [59]. With the hype of web applications, the focus of many researches shifted toward providing authorization-based access control on the web. Damiani et al. [61] focused on developing an access control model for restricting access to web documents using XML format. Their work provides a great understanding of the required concepts but was mostly limited to simple web documents retrieved by a remote user from a server. Later, Role-Based Access Control (RBAC) was introduced to address these problems by changing the underlying subject-object model. RBAC regulates users' access to the information on the

basis of the user's defined roles in the system. RBAC offer an attractive alternative to the strict rigidity of classical MAC, while providing some of the flexibility inherent in DAC [57]. RBAC is simple, reflects organizational structure, and is easy to administer and review. However, it is still difficult and costly to build a good RBAC instance, and a pure RBAC system lacks flexibility to efficiently adapt to changes. Particularly, it is impractical to manually make (and maintain) user to role assignment and role to permission assignment for a dynamic application or a large-scale application with a large number of users or objects.

DAC, MAC, and RBAC models are all identity-based access control models (IBAC) where subjects and objects are identified by unique names and access control is based on the identity of the subject, either directly or through roles assigned to subjects. As a result, IBAC's are most effective for closed and relatively unchangeable distributed systems that deal only with a set of known users who access a set of known services [62]. However, modern cloud environment is far from these rigid assumptions; it is very dynamic with subjects and resources continually added and removed.

With the development of Internet-based distributed systems in late 1990s, a new access model – the Attribute-Based Access Control (ABAC)– appeared. ABAC defines access rights based on attributes of the requester and resource, and users need not to be known by the resource before sending a request [61]. These attributes can be static or dynamic and this makes ABAC popular for Web Services [62]. ABAC is emerging as a dominant form of access control due to its policy-neutral nature (that is the ability to express different kinds of access control policies including DAC, MAC and RBAC) and dynamic decision-making capabilities [63]. Not surprisingly, The US National Institute of Standards and Technology (NIST) recognizes ABAC as a suitable choice for large and federated enterprises over other existing access control mechanism due to its unprecedented amount of flexibility and security [64]. It is straightforward to use ABAC to represent policy based on the attributes of users, objects, and the access environment, and it is easy to revise policy to adapt to a changing application. However, ABAC is typically more complex than classical access controls models (DAC, MAC, RBAC) with respect to attribute definition, attribute relation definition and policy review [65].

With ABAC systems being widely used in service oriented architecture (SOA) based applications, specifications related to ABAC are published as access control-related web services standards that supports ABAC. The standards are expressed in the eXtensible Access Control Markup Language (XACML) and the Security Assertion Markup Language (SAML) [62]. At its core, XACML defines the syntax for a policy language and the semantics for processing policies [66] while SAML defines a framework for exchanging security information such as authentication and authorization decisions in XML format between remote systems.

ABAC's XACML is flexible and scalable and, thus, suitable for cloud computing [62]. Its open standard status, definition in XML, and availability of open source projects has already drawn support from diverse applications. XACML's ability to tie into other authorization systems makes it a natural inter-operability point, even for legacy systems. Its expressive semantics and extensible nature also make it useful as an intermediary language [66]. However, there are practical limitation with using ABAC and XACML in distributed environments such as cloud. It has already been shown that the safety problem of an ABAC system with infinite value domain of attributes is undecidable [63]. Lorch et al. [66] summarizes other problems of XACML as significantly greater size of policies and privilege statements due to the XML encoding overhead and verbosity of the language. XACML does not standardize a complete authorization solution but instead provides a foundation upon which cooperative solutions can manage. Lorch et al. [66] studied different attempts to implement ABAC using XACML within distributed systems and concluded that XACML's flexibility and expressiveness comes at the cost of complexity and verbosity. Moreover, ABAC's XACML lacks from natural support of multiple independent policies that is a requirement in heterogeneous environments such as cloud. Despite XACML's support of arbitrary sub-policies, it is intrinsically designed to supports only single policy. In order to be able to use ABAC for grid computing, Lang et al. [62] extended the access control model and authorization architecture of XACML to encapsulate and support multiple heterogeneous policies without requiring any changes to the policy description and evaluation mechanism of different grid system domains. However, their solution requires making a call to each domain's policy manager for every action in the system. Remote call to individual policy managers for every action is extremely expensive and reduces the performance of the whole system. Cha et al. [67] investigate development of an ABAC framework that supports multiple policies with primary focuses on the cloud environment. Their focus is on providing ways to identify users by their characteristics and attributes rather than predefined identities. Recent efforts in terms of developing a practical distributed authorization system primarily focused on combining different access control models to overcome the limitations of each. Ni and Bertino [68] proposed a new access control language of xfACL to combine the benefits of ABAC's XACML with RBAC. However, their focus is on the language specification rather than implementation and practicality. Huang et al. [65] also propose a framework for combining best features of RBAC and ABAC to provide effective access control for distributed and changing applications. Their model looks as an RBAC from outside while it is internally defined using ABAC. However, they still followed the traditional view of the cloud as data-focused service provider with users seeking data-access services from outside domains.

In summary, although XACML has the potential to be extended to combine results of

multiple policies [69], those policies must be consistent, written with respect to the same access control model, and follow many prior-arrangements. In practice, this large set of prior-agreements prevents XACML from natively supporting multiple independent policies and extensive modifications and extensions are needed to make XACML support multiple policies. This might be an option in applications that services are provided at different domains where a uniform run-time environment to mask the heterogeneity and federation of the underlying system can be reached, e.g. grid computing. However, in our mobile-hybrid-cloud application development framework, hybrid cloud model is a set of public and private cloud resources defined dynamically at run-time. Moreover, the end-user has the option of modifying the pre-defined policy in order to address a certain privacy concern in terms of how different application data are outsourced to different cloud spaces. Thus, it is not possible to assume any prior arrangements between different cloud spaces and a new access restriction system that does not rely on prior arrangements is needed. A few studies have focused on protecting the privacy of the outsourced data for the mobile-cloud applications [54, 70, 71, 1] but their suggested solutions are very restrictive and limited to specific applications. Liu et al. [72] study the privacy protection issue in computation offloading and suggest the use of steganography to hide a mobile picture in another reference picture before sending it to cloud space for processing. However, their suggested solution only works for images and a server can do very little with the combined sent image. Moreover, the server can still extract the hidden image from the combined image, although this is not straightforward. Gentry [73, 74] suggest the use of homomorphic encryption that keeps mobile data private but allows cloud to partially process the data without having the decryption key. However, such encryption can only be used in very limited applications dealing with only data and suffers from all the previous mentioned problems of using encryption.

Our framework is built using an actor programming framework; the framework allows the creation of a pseudo-homogeneous development environment by masking the existing underlying heterogeneities of different cloud environments. Our proposed framework then adds the ability to dynamically restrict access to different cloud spaces, control communication between different cloud spaces and manage new application component creation and the interactions between them [39, 40]. Moreover, it simplifies the development of the privacy and access control policies by removing unnecessary complexity from the XACML grammar and allows support of multiple policies based on the requirements of the application organization, developer, and end-user. In order to respond to dynamic environmental changes, our framework supports context-aware policy-based reconfiguration of actors that is inspired by context-aware web applications presented by Chang and Agha [75, 76] and quality-of-service enabled middleware developed by Venkatasubramanian et al [77].

CHAPTER 3

ARCHITECTURE OVERVIEW

In this chapter we present a framework that embodies components we developed to facilitate parallel mobile hybrid cloud application development. Supporting mobile applications with elastic on-demand cloud resources opens door to overcome mobile hardware limitations. However, in order to properly offload application components into cloud space, to ensure satisfaction of varying application requirements and user expectations, and to address the dynamism in environmental conditions the following challenges should be clarified and addressed:

1. How to model and represent hybrid cloud space, mobile-cloud application, and component interactions to allow efficient and dynamic component configuration and distribution.

2. How to derive current and near-future dynamic environmental parameters, resource usage, network availability, and energy consumption with low overhead.

3. How to represent application target goal, define required policies to restrict accesses and communications, limit offloading budget, specify quality of services, and adjust target goal based on user expectation.

4. How to efficiently configure and distribute application components in a way that all different requirements are satisfied.

5. How to apply dynamic environmental, resource, network, and energy information to component configuration and distribution.

In the remaining of this chapter we first discuss how hybrid-cloud space and mobile-cloud applications are represented in our proposed framework to address 1. We will then explain the initial design and current implementation of the framework and use experimental results to support the idea.

3.1 System Representation

In order to formulate application component distribution into an optimization problem to be solved, we need to have a comprehensive mobile hybrid cloud application model. In this section, we clarify our view on cloud, cloud-applications, mobile-cloud applications, and specify underlying assumptions.

3.1.1 Cloud Model

Over time, cloud services have moved from the model of using public cloud spaces to private clouds and recently to the hybrid model combining both [54, 56]. Cloud infrastructure is traditionally provided by large enterprises, thus, referred to as public cloud. However, storing data on third-party machines suffers from potential lack of control and transparency in addition to legal implications [54, 70, 71, 1]. To address this problem, cryptographic methods are usually used to encrypt the data stored in public cloud while decryption keys are only disclosed to authorized-users [78, 79, 80, 55]. However, these solutions inevitably introduce heavy computational overhead for continuous encryption and decryption of data, distribution of decryption keys to authorized users, and management of data when fine-grained data access control is desired [58]. Cryptographic methods do not scale well, have significant overhead, are expensive to maintain, and are often slow, especially in widely geographically distributed environments such as cloud. To address some of these limitations, several techniques, such as attribute-based encryption (ABE) [81, 82], proxy re-encryption and lazy re-encryption [58], and homomorphic encryption techniques [83, 84, 73], have been developed to provide attribute-based encryption while delegating part of the required decryption to the cloud or providing limited in-cloud computations on the ciphered-text (encrypted data) without revealing data content. However, these efforts still have the traditional data-centric view on the cloud computing, focused on storing data and providing services to access the stored data. If data storage is the primary use for the cloud, the required data access control is already mature enough to effectively implement a fine-grained access control system [85, 86].

In modern mobile-cloud applications, resources stored in the cloud contain more than just data. These resources contain part of the application code that results in access operation meaning execution of the code inside the cloud. It is obvious that the certificate-based authorization systems fail to address this type of applications, as the encrypted piece of code within the cloud cannot be executed without decryption and revealing the content to the cloud provider. As a result, companies gradually moved toward building their own private cloud [54, 87, 88]. Storing sensitive data within private cloud aligns with the traditional

on-premise application deployment model where sensitive data resides within the enterprise boundary and is subject to its physical, logical and personnel security and access control policies [56]. However, owning and maintaining private datacenter is usually not as efficient, scalable, reliable, nor elastic as using the public ones offered, supported and maintained by a large third-party company [87, 88, 89]. Thus, in recent years, a combination of both private and public cloud spaces is used that benefits from all the advantages of the public cloud, while keeping the confidential or sensitive data and algorithms in-house [87, 89]. In order to cover different applications, our framework views the cloud as the most general form combining one or multiple private and public cloud spaces. This allows creation of a general elastic hybrid cloud space to address different needs of a specific mobile-cloud application, while different application components can be executed on different cloud resources according to defined restriction policies, user expectations and application requirements.

3.1.2 Mobile-cloud Application Model

Despite having well explored the benefits of cloud computing to enterprise consumers and service providers over past years, its effect on end users, applications and application developers is still not clear [30]. As mentioned in the previous section, traditional data-centric view of the cloud services needs to be replaced with a more general data-computation-centric view. To reach this, the current common client-server-based service-oriented architecture [25, 30], that provides services on data stored in the cloud to external users, needs to be replaced with a more general elastic architecture that dynamically and transparently leverage cloud resources to provide services and support resource limitations on the end-user device. In such an elastic application development environment, components storing data or performing computations are transparently scattered between the private clouds, public clouds, and the end-user device. When such an application is launched, an elasticity manager monitors the environment and resource requirements of the application and makes decisions about where components should be launched and when they should migrate from device to cloud, or from cloud to device, according to environmental parameters and changes in user preferences [30]. Rather than a simple and rigid solution where nearly all processing and storage are either on the device or cloud, an elastic device should have the ability to migrate functionality between the device and cloud. This ability allows the device to adapt to different workloads, performance goals, energy limitations, and network latencies [30]. One important design objective of this modern application development is to build an infrastructure with enabling functions, such as network protocols, secure communication, and

resource management, in a way that the new elastic computing model introduces minimal extra considerations to application developers. To reach this, unnecessary details of distribution and move-around of application components should be masked from the programmers while access to different components and resources is still restricted for different applications or users.

In order to reach maximum level of parallelism and concurrency without the hassle of traditional multi-threading model, modern cloud-based applications avoid using shared memory model that is unnatural for developers and leads to error-prone non-scalable programs [25, 90]. Instead of relying on global variables and shared states, modern cloud-based applications restrict the interaction between various components to communication using messages. This approach to cloud application development aligns perfectly with the concepts of *actor model of computation* [91] that sees distributed components, called actors, as autonomous objects operating concurrently and asynchronously (Figure 3.1). In response to a received message, an actor can make local decisions, create new actors, send more messages, or change its behavior to respond differently to the next received message [92]. Compared to the traditional shared memory model, actors are a better fit for highly dynamic applications operating in open and challenging environments. Actors may be created and destroyed dynamically, they can change their behaviors, and migrate to different physical locations. The model provides *natural concurrency*, *resiliency*, *elasticity*, *decentralization*, *extensibility*, *location transparency*, and *transparent migration* that ease the process of scaling-up or out, which is a critical requirement for cloud-based applications.

3.1.3 Mobile Hybrid Cloud Application Entities

In order to define different entities involved in a mobile hybrid cloud application, we followed guidelines suggested by Special Publication of National Institute of Standards and Technology on security and privacy in cloud computing [93]. We assume the mobile-cloud application to include the following parties: the owner organization, that owns data and governs cloud infrastructure, different programmers inside the owner organization, that develop different applications using cloud resources, and end-users, that use the developed mobile cloud application. In this view, organization provides mobile cloud application as a product or service to be used by external clients. In addition to these entities, there is public cloud provider that is a third party company providing the cloud resources. To run applications, external application users, or end-users for brevity, install the application on their mobile device. Installed application has different components running on the end-user device or

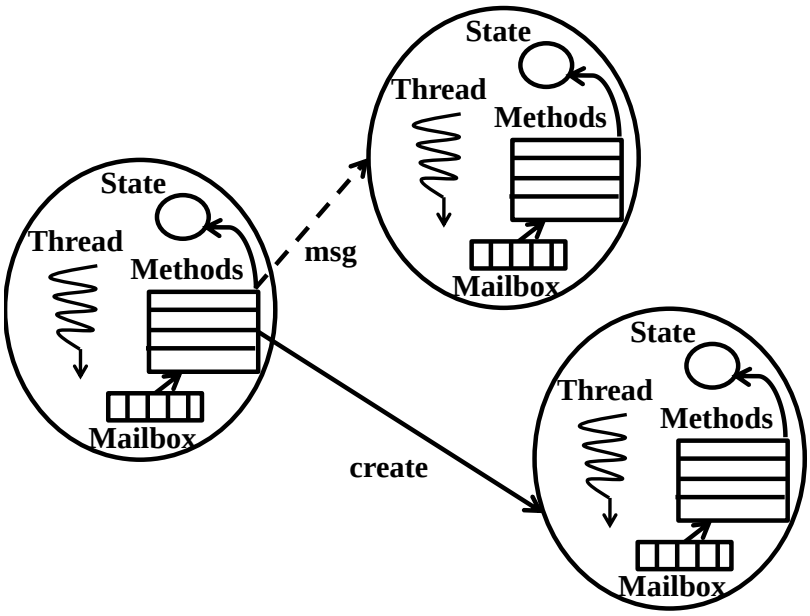


Figure 3.1: Actor model of computation. Actors are concurrent objects that communicate through message-passing and may in turn create new actors. An actor has its own thread of control, a mailbox, and a globally unique immutable name.

provided cloud spaces. Our proposed framework provides solution for automatic dynamic configuration and distribution of these components between user device and different cloud spaces.

3.2 IMCM: Illinois Mobile Cloud Management Middleware Framework

Mobile-cloud computing relies on code offloading process to benefit from available remote cloud servers. Running applications in Virtual Machines (VM) and migrating the entire VM to a more resourceful machine allows benefiting from offloading without processes even knowing of the migration. VM migration is popular because of improving overall performance by running the same code on a more-resourceful machine. With current cloud architecture that runs several virtual machines on one physical machine to provide flexibility and efficiency, system loads between different physical machines can be balanced by migrating VMs out of overloaded/overheated servers. The ability to migrate an entire operating system overcomes difficulties that have traditionally made process-level migration a complex operation [94, 31, 95, 5, 96].

However, VMs are usually large in size (Gigabytes size of data) and migrating them is

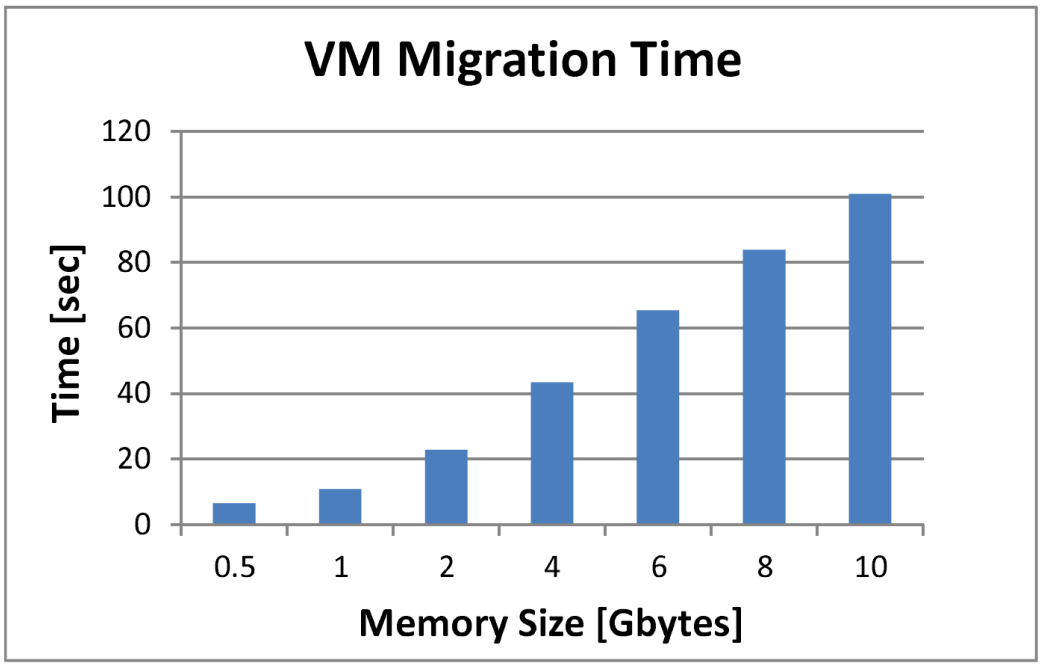


Figure 3.2: Cost of VM migration in a local network. Required time for VM migration in a Local Area Network(LAN). Source: [5]

costly even when performed within a local area network (LAN) [5]. Figure 3.2 [5] shows the result of VM migration time as a function of VM memory size and reveals that it takes on average 10 seconds per 1 Gigabyte of memory for VM migration within a high-speed LAN. Considering that most VMs have a size of more than a few gigabytes, required time for migrating them rests within the order of tens of seconds. However, run-time parameters and program requirements can change in order of milliseconds which requires continuous changes to offloading plan. As a result, VM migration fails to provide a comprehensive solution where fast repeating dynamic offloading is required.

An alternative solution to VM migration, that prevents coarse-grained data transfer, is to migrate application components. As a result, modern offloading processes require decision-making about appropriate parts to offload in addition to migrating them, executing them on remote servers and bringing back the results. Our proposed actor-based mobile-cloud application model provides natural application partitioning and masks component migration process. However, we still need to develop the component offloading decision-making model. In order to develop a code offloading decision-making model for fully parallel mobile applications, we need to specify the target goal for offloading. Offloading goals can vary significantly based on the application or user and range from maximizing the application performance (e.g. games, vision-based applications) to minimizing energy consumption on

the mobile device (e.g. background applications). We have selected two of the most popular offloading goals and developed models for their offloading decision-makings. These two goals include: Maximizing application performance and Minimizing mobile device energy consumption. While offloading to a single remote server and serial monotonic application execution results in similar offloading decision for both mentioned target goals of optimizing for application performance and mobile energy consumption, supporting hybrid cloud environment with multiple public and private cloud spaces in addition to fully parallel application execution results in significantly different component distribution plan for different offloading goals. This requires design and development of two different offloading models. Chapter 4 discusses design and development of an optimal offloading decision-making model with respect to target goal of maximizing application performance and Chapter 5 covers the goal of minimizing mobile energy consumption.

3.3 Evaluation

This section discusses our experimental setup for evaluating our proposed framework. To make the results comparable and link them to our target offloading goal of maximizing application performance, we measure effectiveness as the speedup gained compared to sequential local execution on mobile device. Our selected corpus consists of applications covering different types of programs: CPU intensive, communication intensive, I/O intensive, and combined.

3.3.1 Experimental Setup

Our used equipment include a Samsung Google Nexus S as the mobile device and a Macbook Pro Laptop as the remote offloading server. Table 3.1 summarizes the specifications of our used equipment. Mobile device and the remote server are both on the same WiFi network. We repeated many of our experiments using external cloud spaces and reached similar proportional results. As a result, we include mostly results from tests performed using our own server in the remaining.

The base case in our evaluation is the required time for local sequential execution of the application on the mobile device and the execution speedups are used for comparing different scenarios. In order to account for randomness, we repeat each experiment five times and verify the statistical significance of observed execution times through non-parametric Mann-Whitney U-tests. Unless stated otherwise, the test is two-tailed and the significance level is

Table 3.1: Specifications of the used equipment for evaluation

	Remote Server	**Mobile Device**
System	Macbook Pro-Retina	Samsung Google Nexus S
OS	Mac OSX 10.9.4	Android 4.1.2
VM	JVM (JRE 1.6)	DalvikVM
Processor	Intel Core i7	ARM Coretex-A8
Proc. speed	2.3 GHz	1 GHz
No. of cores	4	1
L2 Cache	256 KB/Core	256 KB
L3 Cache	6MB	-
Memory	16 GB	512 MB

$\alpha = 0.01$.

3.3.2 Program Corpus

Table 3.2 and Table 3.3 list the programs used in the evaluation together with their main characteristics. Evaluation benchmark programs are selected based on their characteristics to cover different application behavior: Computational intensive, Communication intensive, and I/O intensive. In addition, a multi-behavior application is added to combine different characteristics. To avoid a bias towards specific strengths of our approach and to foster comparability, we mostly use similar examples as for works presenting solutions to mobile-cloud computation offloading. The *NQueen* program is a computation-intensive application that places N queens on a $N * N$ chessboard so that no two queens threaten each other [14]. This is a classical puzzle and despite some possible optimization, it still requires checking large number of possible permutations. It is known as a classical computationally intensive problem. We used the SALSA provided implementation for solving this problem that benefits from parallelism and breaks down the board into smaller parts so that they can be processed concurrently. The *Heat* program is a communication-intensive application that simulates heat transfer in a two-dimensional grid in an iterative fashion [5]. Our implementation allows specifying the desired level of communication and both medium and high level of communications are studied. The *Trap* program is a computation-intensive application that calculates a definite integral by approximating the region under the graph as a trapezoid and calculating its area. The *Virus* program reads in file streams from disk and scans for the signature of a given virus [13, 14]. The *Rotate* program is an I/O-intensive application that reads in an image from disk, rotates it in memory and writes it back to disk. Similarly, the *ExSort* program is an I/O-intensive application that sorts the content of a large file using external sort algorithm in limited amount of memory. Finally, the *Image* program combines

all I/O, CPU, and communication characteristics by detecting and recognizing all faces in a given picture using a large dataset of known faces [97, 14, 12, 13]. Its process can be summarized as below:

1. Finding faces in the picture and generating one smaller picture for each detected face.
2. Extracting the features from the picture of each detected face.
3. Performing a similarity calculation against the given database of known faces. This assumes that the features of the database collection are already extracted and finds the minimum Euclidean distance between the given picture and the database.
4. Perform classification and tag all the people recognized in the initial given picture.

Since processing of each picture is performed sequentially, multiple images are processed simultaneously in order to add parallelism.

Table 3.2: Benchmark applications used to evaluate our framework.

Experiment	Description
NQueen	Places N Queens on N*N board
Image	Detects & recognizes all faces in a photo
Trap	Uses trapezoidal rule to calculate definite integral
Virus	Scans a file stream for a specific virus signature
Rotate	Reads, rotates & saves an image to disk
ExSort	External Sort of the content of a file
Heat1	simulates heat exchange on a board
Heat2	simulates heat exchange on a board

Table 3.3: Benchmark application main characteristics showing dominant behavior of the application.

Experiment	Application Characteristic			
	Comp.	Comm.	I/O	
			read	write
NQueen	intensive	-	-	-
Image	intensive	limited	limited	-
Trap	intensive	limited	-	-
Virus	-	-	intensive	-
Rotate	-	-	intensive	intensive
ExSort	intensive	-	intensive	intensive
Heat1	limited	medium	-	-
Heat2	limited	high	-	-

3.3.3 Implementation

Many actor-programming languages have been developed over years to support different applications, including: Erlang [98, 99], ActorFoundry [100], SALSA [101], Scala Actors [102, 103], and Akka [104] and etc. Despite some small differences, most of these programming languages can be used to provide the properties of actor semantics including *encapsulation*, *fair scheduling*, *location transparency*, *locality of references*, and *transparent migration* [105]. Although programmers can use any of these languages to develop cloud-based applications, still the sole practical solution for providing dynamic actor distribution and migration is to code the desired rules inside the developed program as part of the application logic. Our work is focused on removing this additional application logic complexity by separating the actor component distribution management from the logic of the program and developing a dynamic automatic component management. To reach this, we chose SALSA as our programming language mainly due to its loyalty to standard actor semantics. SALSA provides a great support for parallel and distributed programming. Its support for code and data mobility and asynchronous message passing makes programming for distributed systems a natural task. Its coordination model provides an attractive feature for parallel programming where multiple CPUs need to coordinate and communicate between themselves in an efficient manner. SALSA depends on Java, hence it inherits Java's powerful feature of portability across different platforms. All of this makes SALSA an attractive language for mobile-cloud application development [101].

Since SALSA runs on top of Java Virtual Machine (JVM), it can be made to work on Android mobile devices running DalvikVM with some modifications. SALSA provides lightweight actors. The use of lightweight actors makes SALSA highly scalable that is one of the main limitations of some older actor languages. In order to evaluate the performance of actor creation in SALSA, we performed an experiment by creating local anonymous actors, actors created locally without specifying any name or run-time location, local named actors, actors created on the local machine with a specific unique name, and remote actors, named actors created on a remote mobile device connected through WiFi. The results (Figure 3.3) show that SALSA actor creation is significantly fast and it takes less than 100 ms to create a remote actor on a mobile phone connected through WiFi.

A huge advantage of using lightweight actors is the speed and ease of actor migration between different devices. As can be seen in Figure 3.4 it takes less than 200 ms to migrate an actor from a local machine to a remote mobile device working on the same WiFi network. This capability eases the process of mobile application component distribution between different connected spaces.

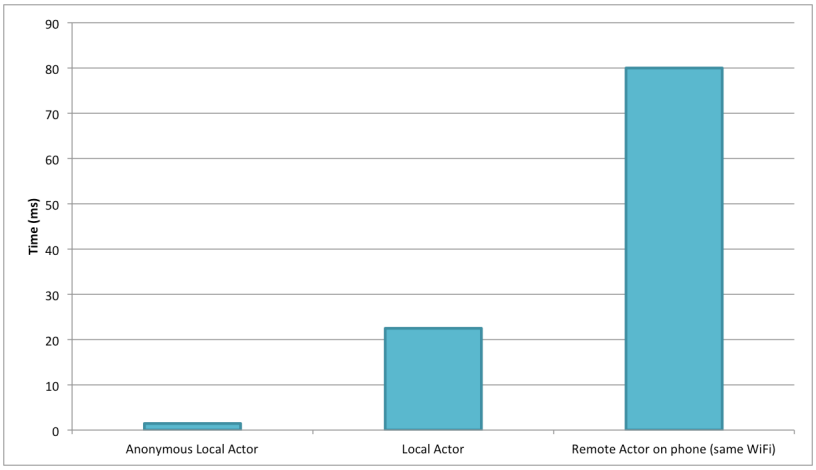


Figure 3.3: Overhead of SALSA actor creation. SALSA actors are lightweight. Both local and remote creation of SALSA actors are significantly fast.

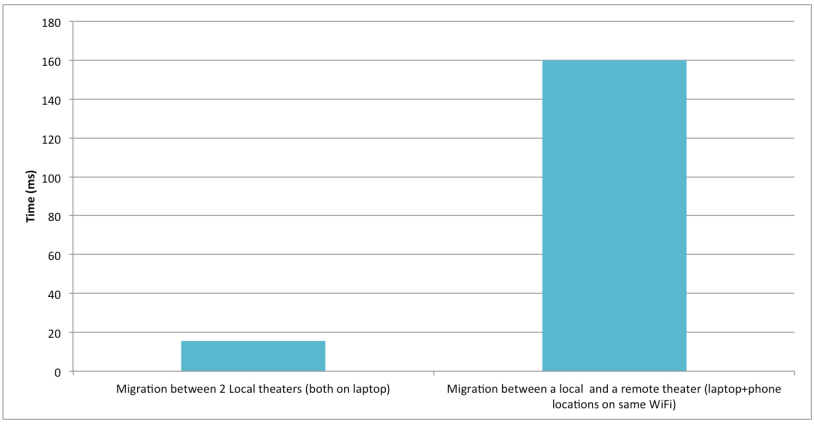


Figure 3.4: Overhead of SALSA actor migration. SALSA actors are lightweight which makes the process of actor migration very fast.

3.3.4 Effectiveness of IMCM in automatic detection of application run-time parameters and offloading appropriate components

In order to support the practicality of the suggested solution, we designed and implemented a simplified elasticity manager for the case of maximizing application performance. Our implementation uses a simplified profiler that records the execution time of each component and reports it to the elasticity manager. Our evaluation is based on the face detection application. As we saw earlier, offloading computation to a more resourceful remote server significantly reduces the total execution time for the face detection application.

Despite significant performance speedup resulting from offloading application to more resourceful systems, manual configuration of components between local mobile device and remote server is not possible. Ideal component distribution depends on several factors that

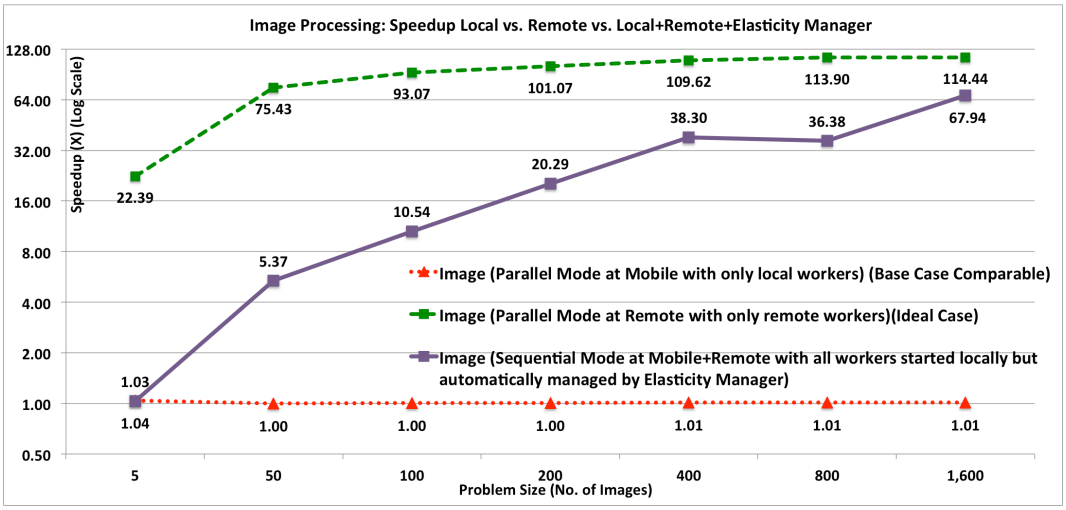


Figure 3.5: Speedup of local, remote, and elasticity manager. Speedup summary for local execution (base case) vs. remote execution (ideal case) vs. local execution with elasticity manager (all automatic management) of image processing problem with different problem size (different number of images to process)

can dynamically change during execution. Thus, an elasticity manager is required to monitor environmental changes and find optimal offloading plan. Figure 3.5 shows the result for manual placement of application components versus automatic component management using IMCM elasticity manager that solves Equation 4.7 and Equation 4.9. Implemented elasticity manager uses the previous profiled execution times of different components at various locations to find the optimal location for placing every component for next interval. We currently do not use profiled execution time from previous execution of the application. Thus, there is an initial lag between start of an application and optimal placement of components resulting from the required time to collect enough profiled data. As a result, when problem size and resulting total application execution time increases, the gap between ideal placement of component and automatic distribution becomes narrower.

3.3.5 Performance overhead of IMCM

While offloading appropriate components to a remote server can potentially improve application performance, having a costly elasticity manager to profile run-time and application parameters and finding optimal distribution plan can result in less overall performance. Figure 3.6 shows the overhead results from our implemented elasticity manager. It shows the performance overhead of the IMCM automatic elasticity manager in practice. Results show that having profiler and elasticity manager running in the background generates $1-5\%$

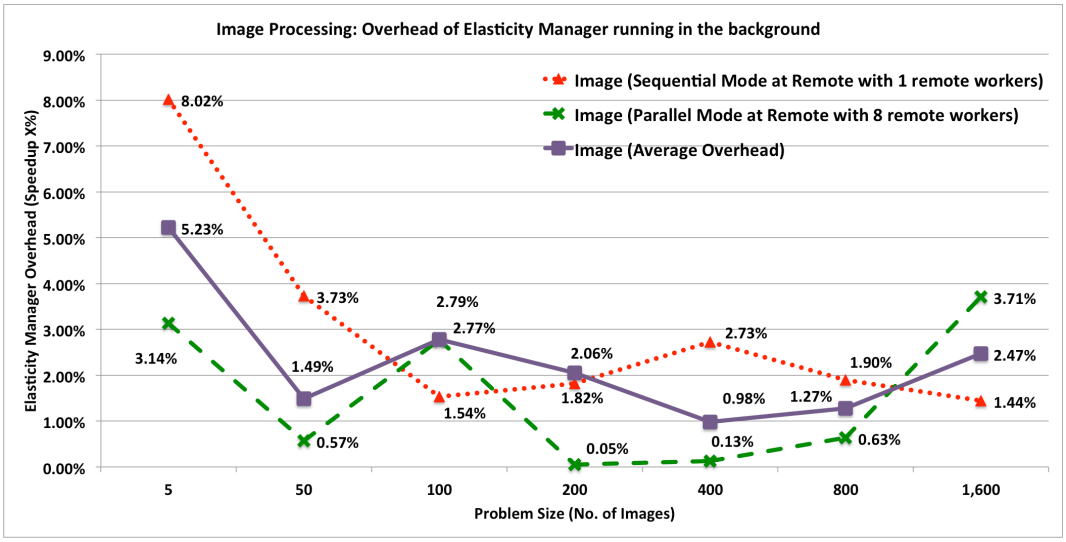


Figure 3.6: Elasticity manager overhead. Overhead resulting from elasticity manager for image processing problem with different problem size (different number of images to process)

speedup decrease on average. Considering the range of $9-60\times$ for speedup gain from offloading applications shows that IMCM elasticity manager overhead is insignificant. Moreover, as the problem size increases, the benefit of offloading becomes more dominant and the elasticity manager overhead becomes even less important.

CHAPTER 4

A DECISION-MAKING MODEL FOR PERFORMANCE-BASED CODE OFFLOADING OF FULLY-PARALLEL MOBILE HYBRID CLOUD APPLICATIONS

While offloading to a single remote server and serial monotonic application execution results in similar offloading decision regardless of the target offloading goal, supporting hybrid cloud environment with multiple public and private cloud spaces in addition to fully parallel application execution result in significantly different component distribution plan depending on the target offloading goal. This requires design and development of different offloading models for various target goals. Offloading goals can vary significantly based on the application or user and can range from maximizing the application performance (e.g. games, vision-based applications) to minimizing energy consumption on the mobile device (e.g. background applications). This chapter discusses our effort toward creating an offloading decision-making model for the target goal of maximizing application performance.

4.1 Offloading Decision for Sequential Applications to Single Remote Server

A straight-froward solution for decision making in terms of offloading an application is to pause execution before processing any part, check on offloading equation and decide whether to offload or not. This section uses this strategy to build a pause-offload-resume decision-making model for sequential applications with a single remote server. We start by initially ignoring the overhead of offloading process and then moving to a more complex solution for both small and large applications.

4.1.1 Code Offloading Decision Ignoring the Offloading Process Overhead

Without considering the offloading process overhead and its effect on application behavior, speedup resulting from running the same code on a more-resourceful machine can be defined as the ratio of available resources on the two machines:

$$Speedup = \frac{S_s}{S_m} = \frac{F_{server} * C_{server} * X_{server}}{F_{mobile} * C_{mobile}} \quad (4.1)$$

where S_s, F_{server}, C_{server}, S_m, F_{mobile}, C_{mobile} are the speed, processor frequency and number of cores of the server and mobile device. X_{server} is the additional speedup resulting from availability of additional resources on the remote server, e.g. caches, memory and potentially more aggressive pipe-lining.

In order to evaluate the speedup achieved by offloading the code from a mobile device to a more powerful remote server in practice, we conducted a series of experiments. Table 4.1 shows the speedup results for our benchmark applications(Table 3.2 and Table 3.3) together with applications' main characteristics.

Table 4.1: Speedup resulting from offloading. Speedup resulting from offloading benchmark applications to a more resource-full remote server. Application characteristic column shows dominant behavior of the application, raw speedup column summarizes maximum speedup gained by running application on a more-resourceful machine excluding offloading overhead, and offload speedup shows maximum speedup resulting from offloading including offloading overhead

Experiment	Application Characteristic				Raw Speedup	Offload Speedup
	Comp.	Comm.	I/O			
			read	write		
NQueen	intensive	-	-	-	73	56
Image	intensive	limited	limited	-	91	44
Trap	intensive	limited	-	-	30	21
Virus	-	-	intensive	-	28	21
Rotate	-	-	intensive	intensive	28	9
ExSort	intensive	-	intensive	intensive	46	36
Heat1	limited	medium	-	-	31	29
Heat2	limited	high	-	-	14	14

While *raw speedup* column ignores the cost of offloading process, *offload speedup* column shows a more realistic view on mobile-cloud offloading by including the required time for offloading process. Note that different rows of the table represents different applications with significantly different behavior, architecture, characteristics, and amount of works that should not be compared with each other. Comparing the values of *raw speedup* and *offload speedup* columns shows the effect of offloading cost on gained speedup. Offloading cost includes the required resources to make offloading decision, offload the application code to remote server and bringing back the result. Ignoring the cost of offloading process, Equation 4.1 predicts the speedup resulting from running the same code on a faster machine. Assuming $X_{server} = 7$ for our experimental setup, the expected speedup is as below:

$$Speedup = \frac{S_s}{S_m} = \frac{2.3 * 4 * 7}{1.0 * 1} = 64 \tag{4.2}$$

raw speedup column of Table 4.1 shows that a speedup of close to 64 times or even higher is possible when offloading overhead is ignored. However, when large amount of data needs to be offloaded (such as *Rotate* application), offloading speedup reached in practice including the offloading process overhead is significantly lower. Moreover, the result highly depends on the application type and behavior as well. A computational-intensive application with high degree of parallelism (e.g. *NQueen*) can benefit from all the additional available resources on the remote server and can reach a high offloading speedup. Extensive I/O operations or communications between different components limits application's ability of benefiting from additional available computational resources at the remote server and reduces the gained speedup (e.g. *Rotate* and *Heat*).

4.1.2 Code Offloading Decision Including Offloading Process Overhead

Equation 4.1 states that offloading is always beneficial, as long as there is a more resourceful server. However, it ignores the required resources for the offloading process and the effect of offloading on application behavior. Only if the required amount of resources for offloading process is small, network connection is fast, and amount of transferred data is small, speedup close to Equation 4.1 can be achieved in practice. For most practical applications, the required resources for offloading process cannot be ignored. As a result, we need to extend Equation 4.1 to include the cost of offloading. Note that we are focused on the target offloading goal of maximizing application performance in this chapter.

Maximizing application performance, or minimizing total execution time, provides real-time applications with higher quality computation in the same amount of time, leading to a smoother and better experience for users. Assuming a small application with w amount of off-loadable work, the goal is to decide whether to offload it or not. Following [1] model, we can summarize the problem as below:

$$\frac{w}{S_m} > \frac{d_i}{B} + \frac{w}{S_s} \quad \rightarrow \quad w * (\frac{1}{S_m} - \frac{1}{S_s}) > \frac{d_i}{B} \tag{4.3}$$

where S_m and S_s are the speed of the mobile device and remote server processors, B is the network connection bandwidth and d_i is the size of data to be transferred. The left side of this equation shows the total required time to execute work w on the mobile device while the right side captures the required time to transfer data to a remote server and execute it

on the server. Obviously, it only makes sense to offload when the left side is larger than the right side. Note that this equation ignores many parameters such as communication latency, required time to bring back the result, etc.

Equation 4.3 can be interpreted in different ways. It is true, if w is large enough, meaning that the program requires heavy computation. It can also be true, if S_s is large, meaning that the server is fast enough. Alternatively, it is true if d_i is low and the required amount of data to be transferred is small. Finally, it can be true, if B is large enough and the available bandwidth is high. In addition to these interpretations, there is also a different way to interpret equation 4.3. It can be seen that in order for offloading to be economical, the following equation must be true:

$$\frac{w}{S_m} > \frac{d_i}{B} \tag{4.4}$$

Equation 4.4 shows that S_s effect is of second degree. In other words, an infinitely fast server ($S_s = \infty$), does not always lead to an offloading decision, if other parameters are not proportional. Only tasks with heavy computation (large w) and small data exchange (small d_i) worth considering for offloading. This shows the need for profiling the performance of different application components in order to detect such tasks [1].

4.1.3 Code Offloading Decision for Large Applications

Use of Equation 4.3 leads to a pause-offload-resume model, where the system pauses before executing any part, checks Equations 4.3 and decides whether to offload or not. If decision is to offload, mobile application will be paused, data transferred to remote server, code executed on remote server, results brought back to the mobile device, and mobile application resumed [12]. However, in communication-intensive applications, offloading single components at a time results in significant remote communications. When components are on the same device, communications are relatively fast and through shared memory space. But when placed on different machines, communications go through multiple network devices and become costly. As a result, components communicating extensively should be offloaded together and the communication cost must be included in offloading decision. The problem of deciding on offloading multiple parts of an application can be formulated into a graph partitioning problem, where nodes are application components, having a weight equal to the amount of their computation, and edges are communications in between, having a weight equal to the amount of transferred data. In such a graph, offloading decision equals finding the minimum cost cut to partition the graph between mobile device and remote server [13, 14]. Note

that application execution is still considered sequential, only one of the components will be executed at any time, and a single remote server is considered for partitioning.

Let's dive deeper in this solution and assume that a mobile application consists of components A, B, C, D, E, and F. Figure 4.1 shows these components as circles, where the size of each circle represents the required amount of computation by that component. Arrows represent communications between different application components and their sizes show the amount of data to be transferred. Table 4.2 shows the relationships between different application components and the size of the communicated data in between. Since we are considering the case of offloading multiple components simultaneously, all the components will be divided between the mobile device and remote server. In order to do this, the minimum cut cost in the graph to partition the components between the mobile device and the remote server should be found.

Table 4.2: Components of a large mobile application and their interactions with each other. Relationship between different components of a mobile application (see Figure 4.1)

Relationship	Size of Input	Size of output
(b,c) = A(a)	a is small	b and c are large
(d,e) = B(b)	b is large	d and e are small
g = C(d,e)	d and e are small	b is small
(f,h,i,j) = D(c)	c is large	f & h are small, i & j are large
k = E(g,h)	g & h are small	k is small
z = F(i,j,k)	k is small, i & j are large	z is small

Assuming that the graph partitioning is performed and resulted in an offloading plan of Figure 4.2, components A, B and C will be executed locally on the phone while components D, E and F will be offloaded to the remote server and executed there. Component A requires medium amount of computation and the size of its input (a) is small. So, it makes sense to execute it locally on the phone. Component B also requires small amount of computation. It requires output from component A that is relatively large amount of data. So, component B should also be executed locally on the phone. Component C requires very small amount of computation and thus, there is no gain in offloading it. Component D requires very large amount of input data (c) and seems to be better executed locally on the phone. However, its required amount of computation is so large that it justifies the transfer of large amount of input data (c) to the remote location and execution of the component there. Component E also requires large amount of computation with small size of input data. Since component D is offloaded, it makes sense to offload E as well. Component F requires medium size of data but since the components generating its required inputs (Components D and E) are already offloaded and the size of its input data (i,j) is also large, it is more efficient to offload and

execute component E on the remote server as well. The final result from component F will then be sent back to the mobile device and returned as the final result of the computation.

4.2 Code offloading and Parallelism

In order to avoid sequential program execution resulting from previous graph-based partitioning approach, CloneCloud [13] and ThinkAir [14] support opportunistic parallelism. When a component is offloaded, the remaining code on the mobile device continues with its execution, as long as the offloaded state is not accessed. As soon as the local code tries to access the state of the offloaded part, local execution is blocked and only resumed when the offloaded code result is received. Despite theoretical potential for parallel execution, this model still leads to sequential execution in practice. In most applications, shared program state is constantly accessed by different parts and mobile code execution remains blocked most of the time. The evaluation results from both CloneCloud and ThinkAir show this phenomena. Those results show similar output for the case of minimizing mobile energy consumption as well as maximizing the performance. Having the same results for both these optimization goals supports the fact that there is not much parallelism achieved in practice. This is one of the main drawbacks of using a shared state program model and a natural result of sequential applications. Moreover, these systems only considers a single remote location for offloading. When parallelism is considered, mobile device and remote server can execute code simultaneously. In addition, when multiple remote servers are considered, application components can be distributed between all of them and be executed at different locations concurrently.

Equation 4.2 predicts the ideal speedup resulting from offloading where computation is large enough, code has high degree of parallelism roughly comparable to available resources, and negligible amount of resources is used for offloading process. Without benefiting from parallelism, running the same code on a more resourceful machine can only provide limited speedup (Sequential remote execution graphs of Figure 4.6 and Figure 4.7). This speedup is mostly because of benefiting from remote server's faster CPU speed, additional available caches, and more memory. However, additional available processing units are not used. We mentioned that for practical applications, the amount of resources required for offloading process is negligible compared to resources required for performing large amount of computation. If computation is not large enough, even using high degree of parallelism does not provide significant additional speedup. However, when the amount of computation is large enough, higher degree of parallelism significantly improves the performance and the benefit

of having additional processing resources becomes visible.

Figure 4.3 shows the relationship between application parallelism degree and speedup resulting from offloading. While on a mobile device with only one core, increasing parallelism degree does not improve the performance, on a more resourceful remote server increasing the program parallelism degree allows better utilization of resources and increases application performance. While sequential execution of *NQueen* problem on a faster system generates a speedup of 14 times, increasing the parallelism degree increases the resulting speedup to 55. N-Queen solution benefits from parallelism by breaking down the problem into smaller parts and executing all parts simultaneously. Thus, different number of workers creates different degree of parallelism for the experiment. Although the number of possible permutations to be investigated increases exponentially when the size of the problem changes from N=8 to N=16, execution time is almost the same when executed locally on the phone regardless of the parallelism degree used. In other words, when executing locally on the mobile phone, using only one solver actor and performing the solution sequentially provides the same result as parallel local execution. This is consistent with the fact that this problem is a computationally intensive application and thus, the total run-time is related to the speed and number of the processors available on the device. If the mobile device used has only one processing core, that core will always be busy with computation and no CPU cycle is wasted. Using any further number of workers provides no additional benefit, as there is no other core to run those solver actors simultaneously. Instead, it even adds the overhead of creating those workers, switching between them, and managing resources among them. *In summary, although N-Queen problem can be solved in parallel, parallelism will not improve total execution time when running the code locally on a mobile device with a single core.*

Studying the cases of remote execution of Figure 4.3 reveals that, on a more resourceful server with multiple cores, using more number of workers improves the performance. Using multiple actors per core gives the best performance, as all cores will be constantly kept busy. Considering the case of sequential execution on the remote server as the base case, we can have a speedup of up to *5* times when executing code in parallel on the same remote server. *So, if enough resources are available on one machine, increasing the level of parallelism can improve the performance.*

Considering the case of remote execution for Nqueen of size=8 in Figure 4.3 shows that there is not much difference in terms of speedup gained between running the problem in a sequential mode versus running it in a full parallel mode when the amount of computation is small. On the other hand, running the case of N=16 on the remote server in sequential or parallel mode makes a huge difference. In fact, when the amount of computation is large enough, the importance of additional cores become much more visible. Running large

amount of computation (N=16 case) on the remote server in the sequential mode provides 2 times speedup, while running the same amount of computation in the fully parallel mode provides 55 times speedup. This means that for large amount of computation, a speedup of 55 times can be reached because of having more caches, faster CPU, more number of cores and existing parallelism in the implemented solution. It also shows that the gain from running the code on a remote server becomes much more significant when the amount of computation increases (N goes from 8 to 16).

Similarly, comparing the local and remote execution of the Image processing application in Figure 4.9 shows that running the face detection algorithm on remote server is roughly more than 200 times faster than running it locally on the phone. Similar to N-Queen problem case, when running on the laptop, we are benefiting from increased amount of caches, increased amount of memory and faster CPU speed. Note that a single image cannot be processed in a parallel mode due to the sequential implementation of the face detection algorithm. In order to evaluate the effect of parallelism on face detection problem, we considered processing more than one image. Since images are processed independently from each other, they can be processed simultaneously and in parallel. The reason for such an improvement when compared to N-queen problem is because of the fact that N-Queen problem requires no I/O operation. However, in face detection problem, images must be read from disk which requires I/O operations. While one worker is loading the image from disk, other workers can work on processing other images. This is in fact the main reason that we see higher speedup for parallel execution on the local mobile device despite the fact that there is only 1 core available on the mobile phone.

Performance improvement resulting from increasing program parallelism degree is limited by the availability of resources. At a certain parallelism degree, resources will become saturated and further increase of parallelism degree will have reverse negative effect (Figure 4.9). Considering the null hypothesis that remote sequential execution is as effective as the remote parallel execution, Mann-Whitney U-test shows that all differences for various problem sizes and parallelism degrees are significant ($P < 0.01$, two-tailed). Consequently, the null hypothesis is rejected.

In order to show the significant effect of parallelism on application performance, we can compare the result from increasing parallelism degree of a problem to the effect of other parameters. A good parameter to study is image quality. Required time for processing an image is related to the size and quality of the image. The larger the image, the more time it takes to process it. Figure 4.4 shows the relationship between image quality and resulting speedup for both local and remote execution. Note that the code for face detection executes sequentially and thus, the remote execution does not benefit from additional number of

cores. It can be seen that *for both local execution on mobile device and remote execution on a more-resourceful server, exponential reduction in image size, reduces the total execution time linearly.* Comparing this with the result from increasing the paralleism degree of a problem (e.g. Figure 4.9) highlights the importance of parallelism degree compared to other parameters. It should also be emphasized the while lowering image quality has significant impact on the accuracy and precision of the results, increasing the parallelism degree has no side effect on accuracy of the results but provides all the benefits of faster processing time.

Although we focused on the goal of maximizing application performance in this chapter, offloading decision for the goal of minimizing mobile device energy consumption is similar for *sequential* applications. As we will see in next chapter, in sequential execution, mobile device remains in idle state consuming energy while waiting for the results from the offloaded code. Consequently, the required time for application execution on the remote server is proportional to mobile device energy usage [12, 13, 14]. However, this effect is limited to sequential applications where only one of the mobile device or remote server executes code at any time.

4.3 Performance-based Offloading Decision Model for Parallel Applications to Hybrid Cloud Environment

Deciding on optimized offloading plan for parallel applications in a hybrid cloud environment requires considering application type, available resources at different remote machines, and the effect of offloading on future application behavior. Similar to previous sections, target offloading goal is maximizing application performance or minimizing total application execution time. We still have a graph G(V,E) where vertices represent application components and edges represent communications in between. The goal is to partition the graph between mobile and different cloud resources in a way that total execution time is minimized. Total execution time consists of the time required to execute the application code in addition to the time required for remote components to communicate and exchange data with each other. Fully parallel execution refers to both parallel execution on multiple remote locations and simultaneous local and remote execution. In other words, mobile device and different cloud spaces execute their components simultaneously. As a result, total application execution time is the maximum time required for any of the mobile or remote spaces to finish executing program code for all of its assigned components. Since local communication between components located on the same machine is relatively fast, we can ignore local communication and only consider communications between different components placed at different

locations. Note that different locations can communicate simultaneously and the total required time for communication is equal to the maximum communication time of different locations. Table 4.3 summarizes notations used in the remaining of this chapter.

Table 4.3: Notations used in parallel offloading model

Notation	**Description**
$B(L)$	Connection bandwidth out of location L
CommAtLoc(L)	Communication time from components on Location L to all other locations
Cores(L)	Number of cores available at Location L
Δ	Time interval of running elasticity manager
Exec(i,l)	Exec. time of component $i \in [1, N]$ at location $l \in [0, M]$
ExecAtLoc(L)	Execution time for all components on Location L
JobCount(i)	Number of requests processed by component i during the time interval Δ
Loc(i,t)	Location of component i at time t
LocAllowed(i, t)	Set of locations at which component i is allowed to be placed at time t. *LocAllowed*$(i, t) \in [0, M]$
$LocEQ(L_1, L_2)$	Checks whether two given locations are identical. Returns 1, if $L_1 = L_2$. Otherwise, returns 0.
MaxAppPerf	Maximum Application Performance
MinAppExec	Minimum Application Execution Time
$ProfComm(i, j)$	Profiled amount of communication between components i and j during the time interval Δ

Using Table 4.3 notations, the offloading goal for parallel mobile hybrid cloud application problem can be summarized as following:

$$\begin{aligned} &max(\ MaxAppPerf\ ) = Min(\ MinAppExec\ ) = \\ &min(\ \max_{0 \le L \le M}(ExecAtLoc(L)\ +\ CommAtLoc(L))\ ) \end{aligned} \tag{4.5}$$

Mobile application consists of N components and each component $i \in [1, N]$ is located at $Loc(i, t)$ at time t. Having M different cloud spaces results in $Loc(i, t) \in [0, M]$ where 0 represents local mobile device and $[1, m]$ corresponds to different cloud spaces. Assuming that we know the application component distribution between the local mobile device and the hybrid cloud spaces at time t_1, our goal is to find optimal component distribution for next time interval t_2 in a way that application performance is maximized. Thus, different parts of Equation 4.5 can be extended as following:

$$ExecAtLoc(L) = \frac{1}{Cores(L)} * \sum_{i=1}^{N} \{LocEQ(L, Loc(i, t_2)) * \ Exec(i, Loc(i, t_2)) * JobCount(i)\} \tag{4.6}$$

Note that both $Exec(i, L)$ and $JobCount(i))$ are provided by the monitoring system and are results of previous profiling of the application. $LocEQ(L_1, L_2)$ considers the execution time of only components running on location L. Similarly, the second part of Equation 4.5 can be extended as below:

$$CommAtLoc(L) = \frac{1}{B(L)} * \sum_{i=1}^{N} \sum_{j=1}^{N} \{LocEQ(L, Loc(i, t_2)) * \ (1 - LocEQ(L, Loc(j, t_2))) * ProfComm(i, j)\} \tag{4.7}$$

As mentioned before, this equation shows the maximum required time for each location to send out all its communications to other locations. $LocEQ(L, Loc(i, t_2))$ considers only components that will be located at Location L at time t_2 and $(1 - LocEQ(L, Loc(j, t_2)))$ captures only remote communications out of location L. Solving these equations results in a set of $Loc(i, t_2)$ that are the optimized locations for different application components during the next time interval Δ. Plugging Equation 4.6 and Equation 4.7 into Equation 4.5 results in the following:

$$\begin{aligned} &max(\ MaxAppPerf\ ) = Min(\ MinAppExec\ ) = \\ &min(\ \max_{0 \le L \le M} (ExecAtLoc(L) \ + \ CommAtLoc(L))\ ) = \\ &min(\ \max_{0 \le L \le M} (\ \frac{1}{Cores(L)} * \sum_{i=1}^{N} \{LocEQ(L, Loc(i, t_2)) * \ Exec(i, Loc(i, t_2)) * JobCount(i)\} \\ &\quad + \frac{1}{B(L)} * \sum_{i=1}^{N} \sum_{j=1}^{N} \{LocEQ(L, Loc(i, t_2)) * \ (1 - LocEQ(L, Loc(j, t_2))) * ProfComm(i, j)\}\ )\ ) \end{aligned} \tag{4.8}$$

It should be noted that equation 4.8 ignores the required time to migrate components between different spaces. This assumption is based on the fact that actor migration is fast

and only transfers actor state and not the actor source code. This is consistent with previous research assumptions that method offloading only takes program state and not the method source codes and experimental results supports its validity for most practical applications.

Since not all components of an application are off-loadable, a few constraints must be added to the above optimization problem to restrict move around of non-off-loadable components. As we are considering a hybrid cloud consisting of multiple private and public cloud spaces, application developers or users can specify additional constraints in terms of how different components can be offloaded to different locations. We will look at the flexibility of the framework in supporting definitions of such restrictions in Chapter 7. These additional constraints can address certain privacy issues in terms of not offloading sensitive or confidential components to public cloud spaces. Required constraints can be expressed as below:

$$
\begin{aligned}
&\textit{subject to constraints:} \\
&Loc(i, t_1) \quad \in \quad LocAllowed(i, t_1) \ : \ \forall \ i \ \in \ [1, N] \\
&Loc(i, t_2) \quad \in \quad LocAllowed(i, t_2) \ : \ \forall \ i \ \in \ [1, N] \\
&\sum_{i=1}^{N} \sum_{j=1}^{N} \{LocEQ(L, Loc(i, t_2)) * (1 - LocEQ(L, Loc(j, t_1)))\} \\
&\quad \leq \ \alpha * Cores(L) \ : \ \forall \ L \ \in \ [0, M]
\end{aligned}
\tag{4.9}
$$

The last constraint is added to prevent flooding too many components at once to a remote server with good initial performance. We limit the number of components that can be offloaded to each remote server to a factor of the number of available cores on that server. α of range 2 to 8 is compatible with our evaluation results, that shows best performance can be achieved when 2 to 8 actors are assigned to each core. If after one round of component move around the target remote server still has enough resources and the execution times are still fast enough, another round of actors can be migrated to that location. In most cases, $LocAllowed(i, t_1) = LocAllowed(i, t_2)$, as the privacy constrained are not often changed during execution. However, the user or the run-time environment has the option of adjusting privacy requirements at run-time whenever needed.

4.4 Experimental Results

There are many factors that can affect an offloading decision ranging from run-time parameters to application type, amount of work, and parallelism degree. In this section we discuss some of these parameters and show our experimental results of their effects on offloading decision.

4.4.1 Effect of run-time parameters on mobile-cloud offloading decision

In order to decide on the beneficiary of offloading w amount of computation to a remote server for our experimental setup, Equation 4.3 can be used with values from table 3.1:

$$w * (\frac{1}{1024Mhz} - \frac{1}{2.3 * 1024MHz * 4 * 7}) > \frac{d_i}{B} \quad \rightarrow \quad B > 1040 * \frac{d_i}{w} \tag{4.10}$$

Rearranging the equation results in $B_{min} \geq 1040 * \frac{d_i}{w}$ to be the minimum required bandwidth in order for offloading decision to reduce total application execution time. The equation depends on the ratio of $\frac{d_i}{w}$ and can only be true when the ratio is small enough. In other words, application offloading is beneficial for large amount of computation (w) and low amount of transferred data (d_i). For values in between, the decision depends on the available bandwidth (B) and an elasticity manager must evaluate the equation based on run-time parameters. Figure 4.5 summarizes equation 4.4 for different amount of computation (w) and communication data (d_i).

For *N-Queen* problem, a single integer value has to be transferred both for input value (N) and final result and d_i is very small. At the same time, problem is computational-intensive and requires large amount of computation (large w). According to Wikipedia [106], for $N = 8$ there are 4,426,165,368 possible arrangements but only 92 solutions. Even using some optimization tricks, the size of possible permutations to be checked is only reduced to 16,777,216, which still requires a brute-force approach to investigate all. Applying equation 4.10 to this problem results in $B_{min} = 1040 * \frac{1}{16} = 65 \frac{bit}{sec}$. This is the minimum required bandwidth to make offloading of the N-Queen problem efficient. Table 4.4 shows average bandwidth for different existing technologies. It shows that any type of network connection provides enough bandwidth and offloading always improves application performance. Note that the code of the N-Queen solver is assumed to be available on the remote server and network latency is ignored. So, as long as there is a reliable Internet connection for the mobile phone, it always makes sense to offload the N-Queen problem computation to the remote server. However, note that we are ignoring the existing latency in sending and re-

ceiving the data. Large values of latency can change the above interpretation. Fortunately, public cloud providers have several geographically distributed datacenters that can provide a relatively-close remote server for most mobile users.

Table 4.4: Average bandwidth of different mobile technologies

Technology	Average Existing Bandwidth (bit/sec)
2G (Edge)	0.2 Mega
3G	1 Mega
4G (LTE)	5 Mega
WiFi	10 Mega

In case of the *Image* problem, assuming remote server to be super fast ($S_s = \infty$), offloading decision depends on w, d_i and B. If detection of faces in the initial image, extracting features for every detected face and comparison to database are all offloaded, the entire initial image needs to be transferred to the remote server and the amount of communicated data (d_i) is large. Thus, it is only beneficial to offload, if B is large enough. On the other hand, if the initial detection of faces are performed locally and only the extracted features are transferred, d_i is much smaller. Consequently, even for slower network connections, offloading of the remaining parts is beneficial. This highlights the importance of considering the combination of all parameters for deciding on offloading. Different parts of an application can become offloading candidates at different time and an elasticity manager is required to dynamically decide on offloading based on run-time parameters.

4.4.2 Effect of application type on mobile-cloud offloading decision

We saw earlier in the effect of offloading process overhead on resulting speedup. However, there is a significant difference between offloading speedup for different applications. One of the main reasons for such a large difference is application type. For applications that require large amount of data to be transferred (such as *Rotate* application), offloading speedup reached in practice is significantly lower. The main reason is that these applications take large amount of data with themselves when offloaded to remote locations and the overhead of offloading becomes very large. On the other hand, applications that has large amount of computation but requires only small amount of data to be transferred (e.g. *NQueen*) can benefit from all the additional available resources on the remote server and can reach a high offloading speedup without spending much resources for offloading process. N-queen problem is a computationally intensive application with high degree of parallelism. So, when executed on a more resourceful system, it can benefit from all available resources and reach a speedup

comparable to equation 4.1. Extensive I/O operations or communications between different components limits application's ability of benefiting from additional available computational resources at the remote server and reduces the gained speedup (e.g. *Rotate* and *Heat*). *Image* application requires reading and writing from disk in addition to processing images and is a combination of both computationally and I/O intensive types. Since I/O operations are part of image processing application, its performance is affected by delays resulting from that and the speedup reached by running it on a more resourceful system is slightly less than equation 4.1. Table 4.1 reveals that applications with intensive computations are the best candidates for offloading. On the other hand, having intensive I/O operations or communications reduces the benefits of offloading. Thus, application type and behavior have significant impact on offloading results and different applications need different offloading plans even in same environmental settings. This highlights the requirement of profiling application component performance in order to gain a real-world perspective on its behavior and real requirement.

4.4.3 Effect of problem size (amount of work) on mobile-cloud offloading

We mentioned earlier that the amount of computation of a problem has significant impact on offloading results. With offloading process overhead having significant impact on resulting speedup, transferring less amount of data and larger computations improves the offloading speedup. Such observation suggests that larger problem sizes can potentially lead to increased offloading speedup. In order to investigate this hypothesis, we conducted a bunch of experiments with different amounts of computations. Figure 4.6 and Figure 4.7 show offloading speedup for different amount of work for *NQueen* and *Image* applications. By comparing the remote execution curves and local execution curve, it is revealed that larger amount of work results in more computationally-intensive applications, reduces the importance of the fixed amount of work required for offloading process, and increases the gained speedup. While initial offloading speedup of *NQueen* problem is almost equal to 1 (for N=8) due to low amount of required computation, changing N value exponentially increases the amount of work to be performed and similarly increases the resulting speedup.

Image problem is a multi-behavior application with initial speedup of larger than 1 due to the size of computations required for processing even one single image. For this problem, changing the amount of work equals increasing the number of images to be processed and results in linear increase of speedup. Having larger number of images to process allows creating a fixed number of worker actors and reuse them for processing other images. This

further lowers the overhead of offloading process by preventing the continuous creation and removal of worker actors. As a result, there is a linear relationship between the number of images and the total execution time for remote execution of the *Image* problem.

4.4.4 Comparison of Sequential Local Application Execution versus Parallel Local and Remote Execution

While offloading computation to a more resourceful system can improve overall application performance, mobile device local resources are wasted while waiting in the idle state for the result of offloaded code to be returned. With mobile devices becoming more powerful, this wasted computational power can be put to a better use. Our proposed framework supports simultaneous local and remote application execution and uses local mobile resources to execute other parts of an application while waiting for the offloaded code result.

Figure 4.8 shows the speedup differences between processing different number of images using only remote server and simultaneous execution on both local device and remote server. Since processing of a single image is sequential, for small amount of work (small number of pictures to process), total execution time will be dominated by the required time for local mobile device to process its share. This will result in remote server starvation and waste of resources, as there will be no more job for it to process. However, with increase in the amount of work, there will always be enough job for remote server to perform and the advantage of using both local and remote server for application code execution becomes visible.

Figure 4.9 shows the same effect based on application parallelism degree. We mentioned earlier that higher degree of parallelism will increase the flexibility of the application and results in higher offloading speedup. However, this is only true, if enough computational resources are available. As can be seen in the graph, increasing the parallelism degree (number of workers) initially results in higher speedup but after a certain point this effect is reversed. In fact, having higher degree of parallelism than the available resources results in over-saturation of resources, adds the overhead of managing all those workers, and reduces overall speedup. Our results show that required parallelism degree for an application to reach highest speedup is proportional to number of processing cores available. The coverage differences of any two different number of workers for both remote and simultaneous local and remote executions are significant ($\alpha = 0.01$). Thus, the null hypothesis that there is no significant difference between image processing execution with different number of workers can be rejected.

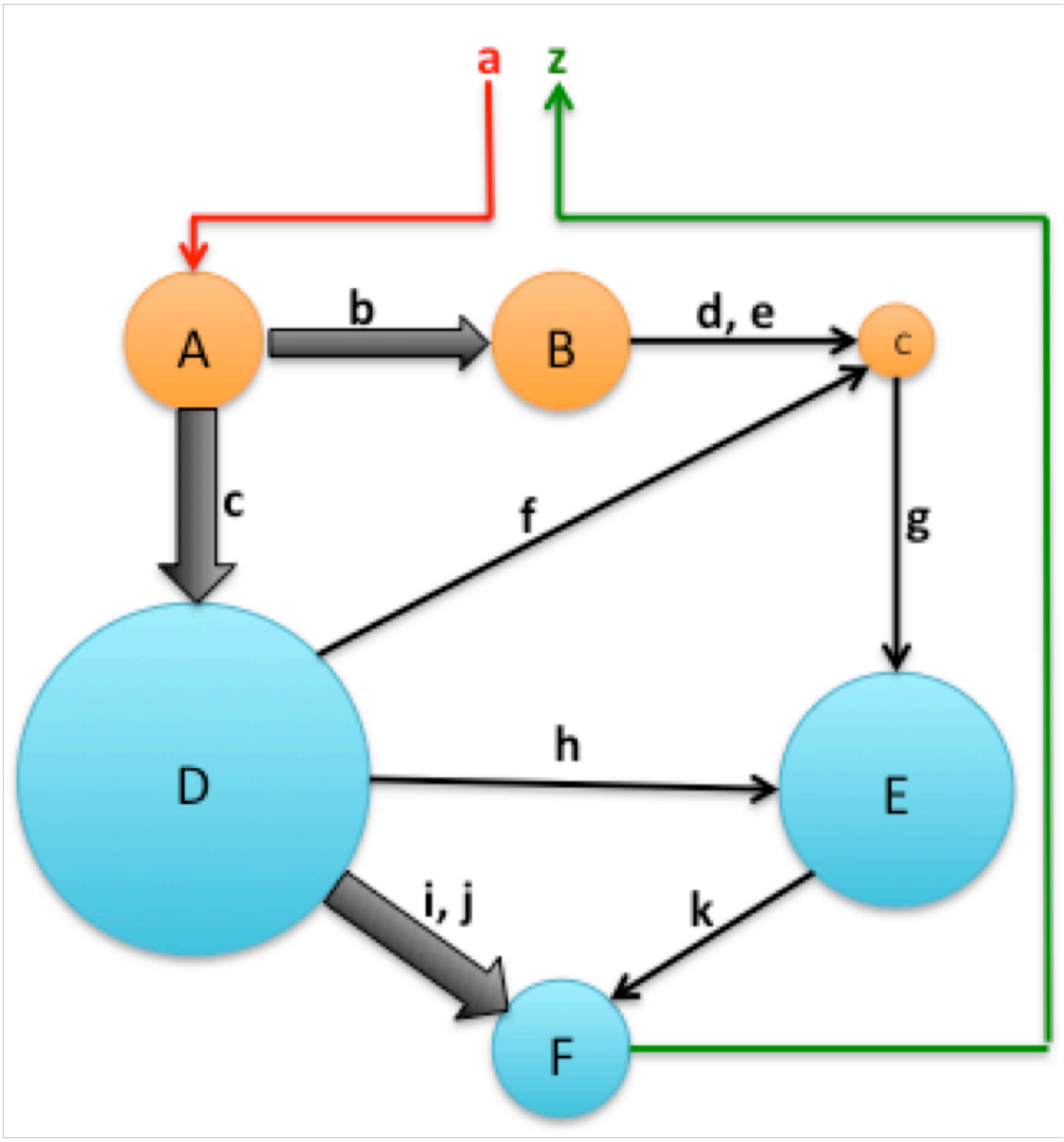


Figure 4.1: A mobile Application represented as a graph before partitioning. Application components are represented as circles where their sizes represent required amount of computation. Arrows represent communication between different components where arrow sizes shows the amount of data to be transferred.

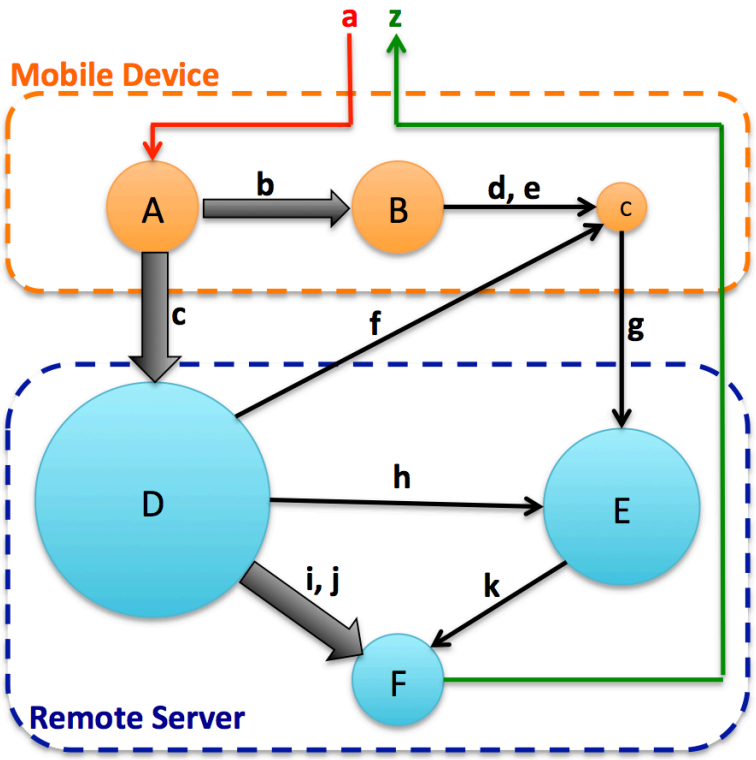


Figure 4.2: A mobile Application represented as a graph after partitioning. Application components are distributed between mobile device and the remote server.

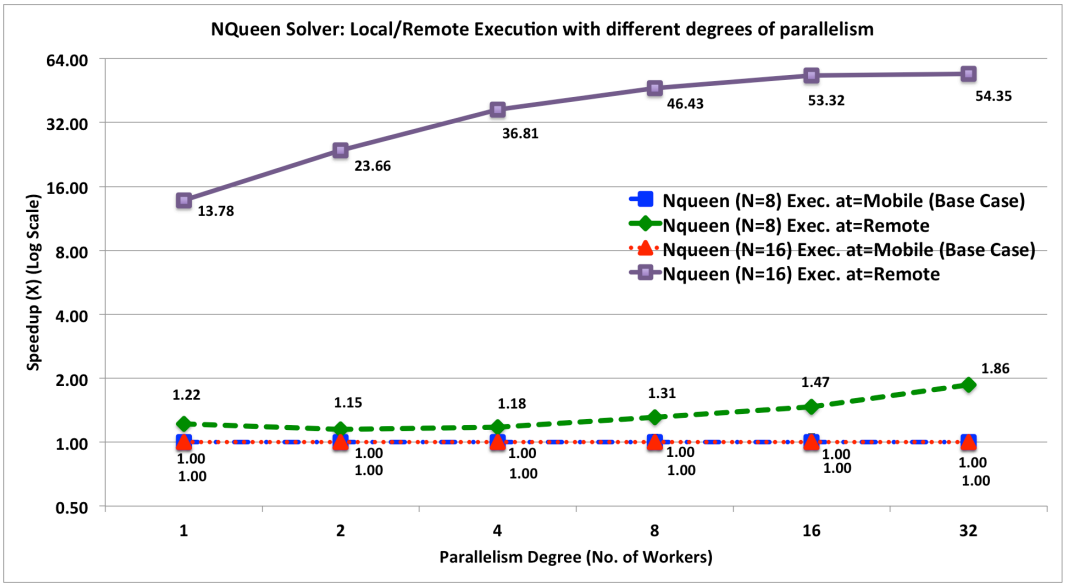


Figure 4.3: Speedup for NQueen problem. Speedup summary for local and remote execution of NQueen problem with different degree of parallelism

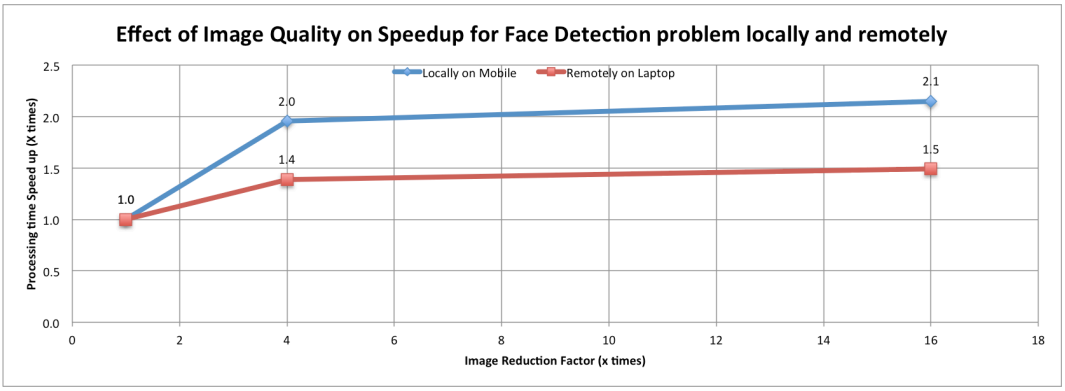


Figure 4.4: Effect of Image Quality on Speedup for Face Detection algorithm. When processed locally on the mobile device and remotely on a more resourceful server

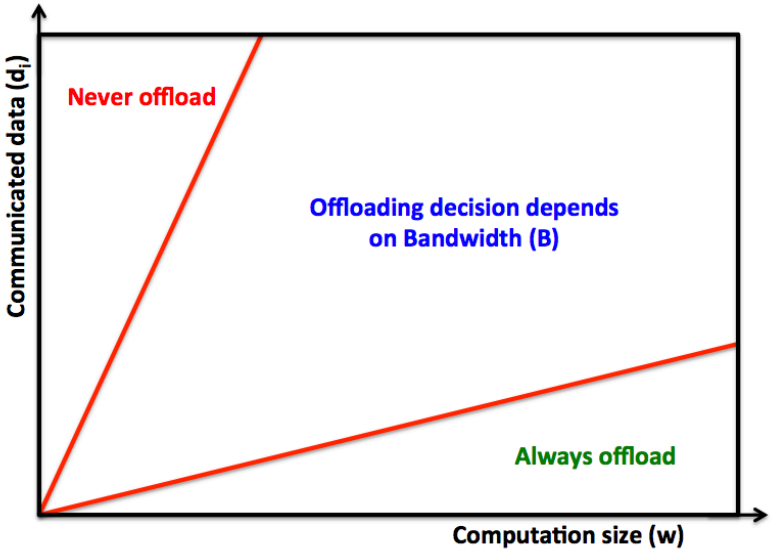


Figure 4.5: Offloading decision for optimizing application performance. A combination of to-be-transferred data size, computation size and bandwidth determines whether offloading is beneficial or not. Source: [25]

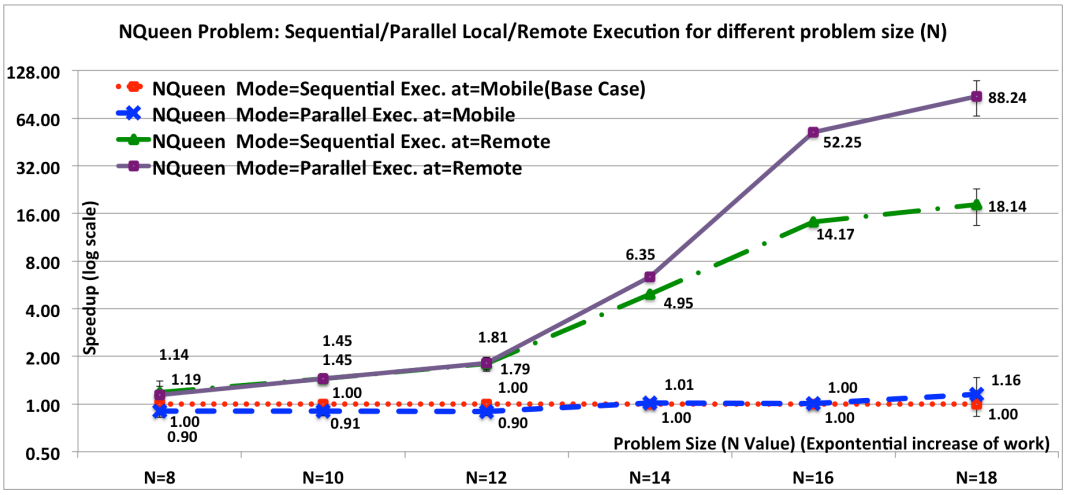


Figure 4.6: N-Queen problem with different amount of work. Speedup summary for local and remote execution of N-Queen execution for different amount of work (different problem size)

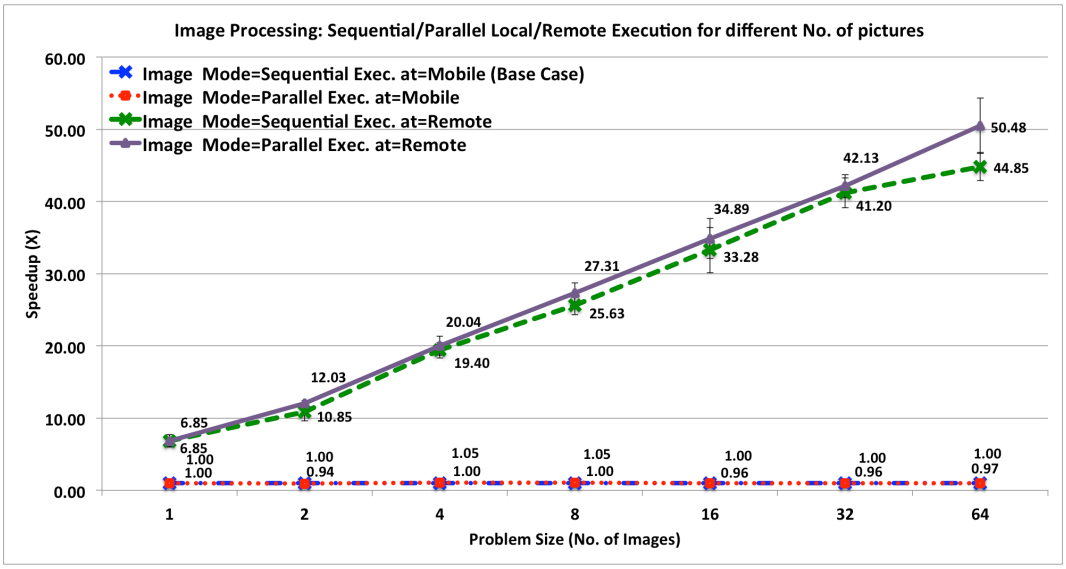


Figure 4.7: Effect of amount of work on speedup. Speedup summary for local and remote execution of Image Processing application for different amount of work (different no. of images to process)

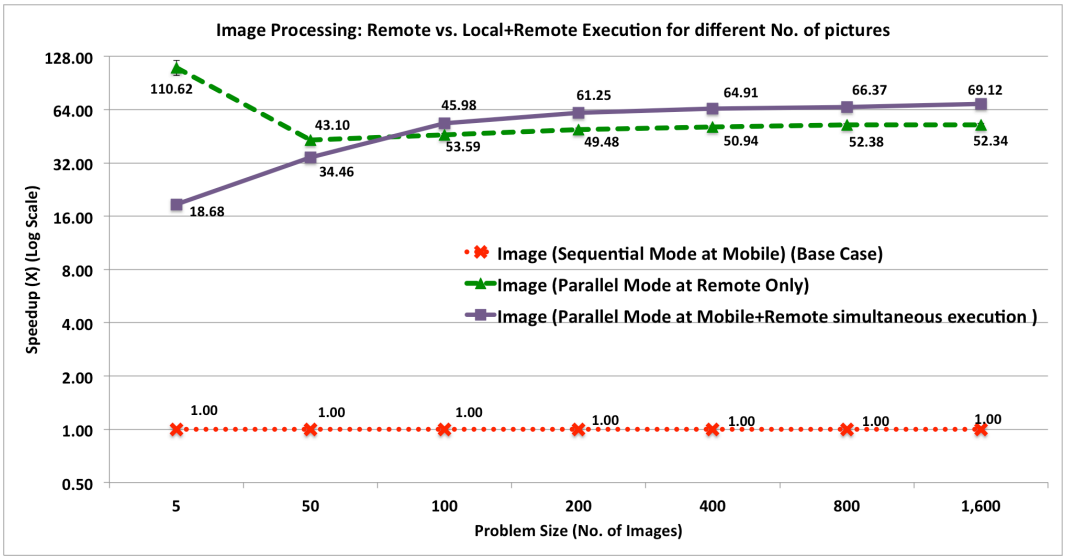


Figure 4.8: Speedup different problem size. Speedup summary for remote execution vs. local+remote execution of image processing problem with different problem size (different number of images)

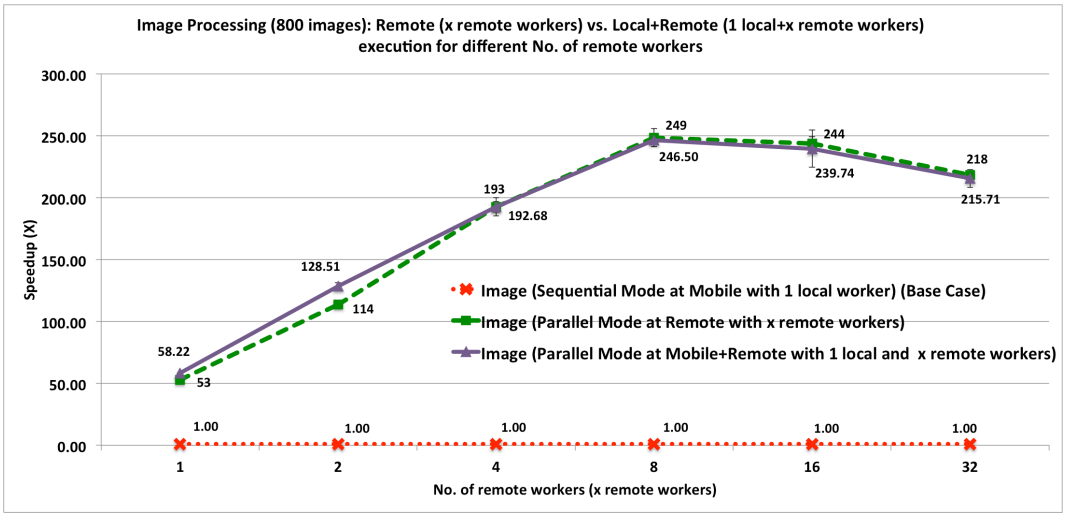


Figure 4.9: Remote vs. local+remote execution. Speedup summary for remote execution (x remote workers) vs. local+remote execution (1 local + x remote workers) of image processing problem with different number of remote workers

CHAPTER 5

A DECISION-MAKING MODEL FOR ENERGY-BASED CODE OFFLOADING

Chapter 4 studied an offloading decision model for maximizing application performance. We also mentioned that for sequential application execution with a single remote server, the result of offloading decisions model is the very similar regardless of target offloading goal. Unless the device goes into deep sleep mode while waiting for execution of the offloaded code, its energy consumption is proportional to the waiting time. In this chapter, we focus on the target offloading goal of minimizing mobile energy consumption and create an offloading decision-making model for this target goal.

5.1 Offloading Decision for Sequential Applications to Single Remote Server

In order to build an offloading decision model for sequential execution with a single remote server, we follow an approach similar to Chapter 4. We start by initially ignoring the overhead of offloading process and then move toward a more complex solution for both small and large applications.

5.1.1 Energy-based Offloading Decision Model for Sequential Applications Ignoring Offloading Process Overhead

Ignoring the overhead of offloading process, Equation 4.1 is valid regardless of the target offloading goal. Table 5.1 shows the energy saving ratio results for our benchmark applications(Table 3.2 and Table 3.3) together with applications' main characteristics. Energy saving ratio (ESR) is defined as the ratio of saved energy compared to the energy consumed by local sequential execution on the mobile device.

Table 5.1: Energy saving ratio resulting from offloading. Energy saving ratio resulting from offloading benchmark applications to a more resource-full remote server. Application characteristic column shows dominant behavior of the application, raw ESR column summarizes maximum energy saving ratio gained by running application on a more-resourceful machine excluding offloading overhead, and offload ESR shows maximum energy saving ratio resulting from offloading including offloading overhead

Experiment	Application Characteristic				Raw ESR	Offload ESR
	Comp.	Comm.	I/O			
			read	write		
NQueen	intensive	-	-	-	-	-
Image	intensive	limited	limited	-	91	44
Trap	intensive	limited	-	-	-	-
Virus	-	-	intensive	-	-	-
Rotate	-	-	intensive	intensive	-	-
ExSort	intensive	-	intensive	intensive	-	-
Heat1	limited	medium	-	-	-	-
Heat2	limited	high	-	-	-	-

5.2 Energy-based Offloading Decision Model for Sequential Applications Including Offloading Process Overhead

Code offloading in order to minimize mobile energy consumption is popular goal. Using less mobile energy allows mobile devices to last longer without requiring charging. This is of prime importance for service or background applications that runs in the background for a long time. An equation similar to 4.3 can be formed to decide on the benefit of code offloading for w amount of work when the goal is to minimize energy consumption on the phone for sequential execution [1]:

$$P_1 * \frac{w}{S_m} > P_2 * \frac{d_i}{B} + P_3 * \frac{w}{S_s} \rightarrow w * (\frac{P_1}{S_m} - \frac{P_3}{S_s}) > P_2 * \frac{d_i}{B} \tag{5.1}$$

where P_1, P_2 and P_3 are the power consumption of the mobile device when performing computation at the highest speed, communicating data over network, or in the idle mode. S_m and S_s are the speed of the mobile device and remote server processors, B is the network connection bandwidth and d_i is the size of data to be transferred. The left hand side of Equation 5.1 shows the amount of energy consumed on the phone when performing the computation locally while right hand side captures the amount of energy required to transfer input data to the remote server in addition to energy consumed by the mobile device while waiting for the offloaded code result. *It should be noted that the focus is on minimizing the amount of energy consumed on the phone and not the total amount of energy required for the computation.*

Equations 5.1 and 4.3 are very similar and usually results in close decisions, if P_1, P_2 and P_3 are proportional. This is usually the case, as the mobile screen is assumed to be on even in idle state. Turning screen off can definitely help save energy but it will be an uncomfortable experience for the users to see the screen going on and off several times during the execution of an application. As a result, all previous research [12, 13, 14] assumed the mobile screen to remain on even in idle mode.

This is , in fact, the main reason that all the previous research [12, 13, 14] reached the same offloading decision plan regardless of optimizing for maximizing application performance or minimizing mobile energy consumption. As mentioned before, all those research resulted in a sequential application execution even when, in theory, their models support opportunistic parallel execution. In fact, method-level offloading results in a pause-offload-resume model, where the system pauses before executing any method, checks either of the equations 5.1 or 4.3 and decides whether to offload or not. If the decision is to offload, the phone application will be paused, data transferred to remote server, code executed on remote server, results brought back to the phone, and the phone application resumed. MAUI [12] was the first to fully implement this approach at method level and showed that it can help both save energy on the mobile device and improve the performance of the application.

5.3 Energy-based Offloading Decision Model for Parallel Applications to Hybrid Cloud Environment

Fully parallel execution refers to both parallel execution on multiple remote locations and simultaneous local and remote execution. In order to decide on optimal offloading plan for minimizing mobile energy consumption in fully parallel execution model, we need to develop a model that includes different offloading locations in addition to the effect of component distribution on application internal behavior. In this section we develop a model for this purpose. The goal of minimizing mobile battery energy consumption can be extended as below:

$$
\begin{aligned}
&min(\textit{Application Mobile Energy Consumption}) = \\
&max(\textit{Energy Saving on Mobile Device}) = \\
&max(\textit{Total Mobile Energy Saving by remote comp. exec.} \\
&\qquad - \textit{Energy Loss due to local comm. becoming remote comm.} \\
&\qquad + \textit{Energy Save due to remote comm. becoming local comm.} \\
&\quad)
\end{aligned}
\tag{5.2}
$$

The above extension is based on the fact that all parts of an application has to be executed locally on the phone, if no offloading is made. The first part of equation 5.2 can be further extended as below:

$$
\begin{aligned}
&\textit{Total Mobile Energy Saving by remote comp. exec.} = \\
&\qquad \sum_{i=1}^{N} (\ \textit{LocEQ}(0, \textit{Loc}(i, t_1)) * (1 - \textit{LocEQ}(0, \textit{Loc}(i, t_2))) \ \ * \textit{Energy}(i)\ )
\end{aligned}
\tag{5.3}
$$

where $\textit{Energy}(i)$ is the profiled energy consumption of component i running locally on the mobile device during the time interval Δ. Note that the first term of the equation considers only components that are currently on the phone and second term adds the condition that those element must now be at a remote location. This way energy saving is only counted for components that have been migrated from the local phone to a remote location. It should be noted again that our goal is to minimize energy consumption at the mobile device and not the total energy. Thus, the migration of components between remote locations does not help with this goal and is not considered in the equation.

The second part of equation 5.2 can also be extended as below:

$$
\begin{aligned}
&\textit{Energy Loss due to local comm. become remote comm.} = \\
&\sum_{i=1}^{N} (\ \textit{LocEQ}(0, \textit{Loc}(i, t_2)) * (1 - \textit{LocEQ}(0, \textit{Loc}(j, t_2))) \ * \textit{ProfiledComm}(i, j) * P_{\textit{commMobile}}\ )
\end{aligned}
\tag{5.4}
$$

where $\textit{ProfiledComm}(i, j)$ is the profiled amount of communication between components i

and j during the time interval Δ and $P_{commMobile}$ is the mobile power when communicating. Note that first term of the equation restrict the summation to components i that will be local on the phone while the second part limits the problem to components j that will be on a remote location. As a result, the combined effect it to limit the summation to communication between components that one will be local on the phone and the other one is in a remote location.

Similarly, the last part of equation 5.2 can be extended as below:

$$\begin{aligned} &\textit{Energy Save due to remote comm. become local comm.} = \\ &\sum_{i=1}^{N}(\ \textit{LocEQ}(0, \textit{Loc}(i, t_1)) * (1 - \textit{LocEQ}(0, \textit{Loc}(j, t_1))) \ * \textit{ProfiledComm}(i, j) * P_{commMobile} \ ) \end{aligned} \tag{5.5}$$

this will result in the energy saving from components that were previously remote but are now local on the phone and communicate locally. Note that we are assuming that local communication is very efficient and ignoring its energy consumption. This is a valid approximation considering the fact that most local communication is through shared memory space or reference passing rather than sending the message through network devices.

In terms of required constraints, we have the following restrictions:

$$\begin{aligned} &\textit{subject to}: \\ &\qquad \textit{Loc}(i, t_1) \in \textit{LocAllowed}(i, t_1) \ : \ \forall i \in [1, N] \\ &\qquad \textit{Loc}(i, t_2) \in \textit{LocAllowed}(i, t_2) \ : \ \forall i \in [1, N] \\ &\textit{Total Execution time after offloading} \leq \\ &\qquad \beta * \textit{(Total exec. time for running all comp. locally on phone)} \end{aligned} \tag{5.6}$$

The last constraint guarantees that by offloading components to remote locations to save local energy, we are not affecting the performance of the application in a way that it becomes unacceptable. In other words, it allows energy saving as long as a certain service performance quality is satisfied. The range for β varies from 1.05 to 1.10 for most practical applications. This is one of the parameters that can be adjusted based on application developer or user expectations. This restriction can further be extended similar to equations 4.6 and 4.7 as below:

$$\max_{0 \le L \le M} \{ \frac{1}{Cores(L)} * \sum_{i=1}^{N} (LocEQ(L, Loc(i, t_2)) * Exec(i, Loc(i, t_2)) * JobCount(i)) + \frac{1}{B(L)} * \sum_{i=1}^{N} \sum_{j=1}^{N} (LocEQ(L, Loc(i, t_2)) * (1 - LocEQ(L, Loc(j, t_2))) * profiledComm(i, j)) \} \le \beta * \frac{1}{Cores(L)} \sum_{i=1}^{N} Exec(i, 0)) \quad (5.7)$$

As mentioned, this last constraint adds some limitation in performance reduction because of code offloading. This was not a big factor in many of the previous research, as their formulation based on sequential code gave almost same result for offloading for both energy saving and performance improvement goals. However, since we are using a full parallel model, the results from the two optimization goals are very different. This is the main reason that we added some restriction on how much improving for one goal can affect the other. A similar restriction can also be added for the case of performance optimization where we want to limit the mobile energy consumption increase to a certain level.

A big challenge to solving Equation 5.3 is the use of *Energy*(i). As mentioned, *Energy*(i) is the profiled energy consumption of component i running locally on the mobile device. This requires fine-grained profiling of energy consumption per application component on mobile device. However, most mobile devices do not provide any tool for direct measurement of the consumed energy. Almost all previous research in this area rely on external power meters to measure energy consumption. Although using expensive external power meters work for experimental settings, we cannot expect end users to carry such a device with themselves to profile energy consumption of the mobile device. This is a big challenge for optimizing energy consumption of mobile hybrid cloud applications. Even if the total energy consumption of the mobile device can be measured, there are multiple applications running on a mobile device at any time. This requires distribution of the total measured energy among those applications, which itself is another challenge. Even if this can solved, there are multiple components within our target application running over times and distributing energy further among those application components is another big remaining issue. We discuss these challenges as part of our online monitoring and energy modeling system in Chapter 6.

5.4 Experimental Results

In order to evaluate our assumption in building an energy-based offloading decision-making mode, we performed several experiments. The remainder of this section discuss the results of those experiments.

5.4.1 Effect of application type on mobile-cloud offloading decision

Ignoring the cost of offloading and internal communication between different application components, offloading always results in energy saving on mobile device as long as the remote server is faster than mobile device. Offloading cost includes the required resources to make offloading decision, offload the application code to remote server and bringing back the result. Different applications have significantly different behavior, architecture and characteristics that can significantly affect the offloading decision. While *NQueen* components need to take very limited input data to a remote location, *Rotate* application requires taking the entire picture. As a result, energy used by mobile device to offload these components can become larger than required energy for computation making offloading ineffective. The result highly depends on the application type and behavior. A computational-intensive application with high degree of parallelism (e.g. *NQueen*) can benefit from all the additional available resources on the remote server and can reach a high offloading speedup. Extensive I/O operations or communications between different components limits application's ability of benefiting from additional available computational resources at the remote server and reduces the gained speedup (e.g. *Rotate* and *Heat*).

5.4.2 Effect of problem size (amount of work) on energy-based mobile-cloud offloading

Figure 5.1, Figure 5.2, and Figure 5.3 show mobile energy usage, energy saving ratio, and execution speedup for different amount of work for *Image* application.

5.4.3 Effect of application parallelism degree on energy-based mobile-cloud offloading

Figure 5.4, Figure 5.5, and Figure 5.6 shows the effect of application parallelism degree on mobile energy usage, mobile energy saving ratio and execution speedup resulting from offloading.

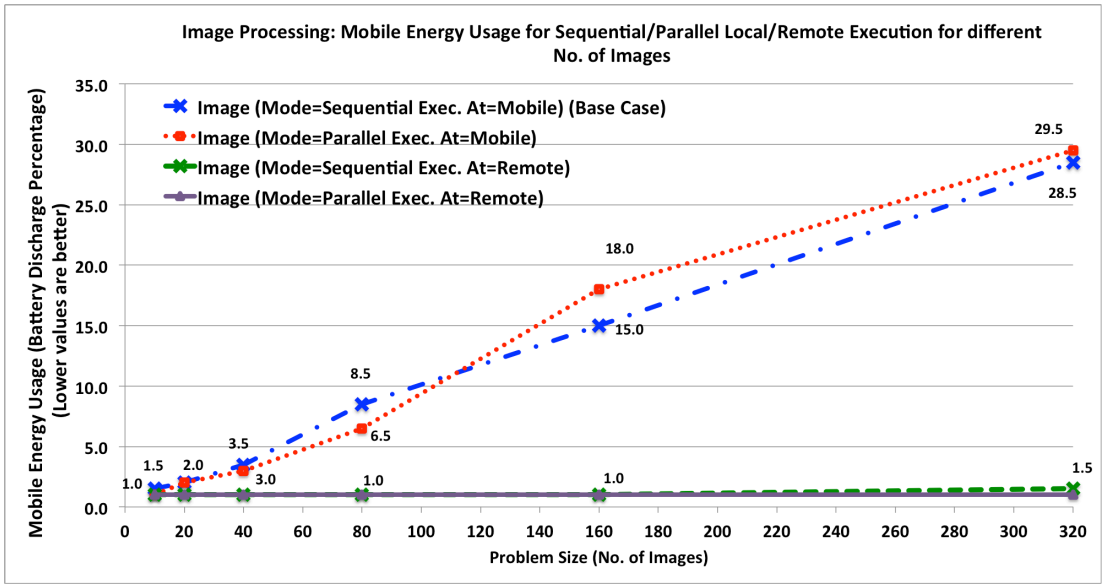


Figure 5.1: Energy usage with different amount of work. Mobile energy usage for local and remote execution of Image Processing application for different amount of work (different no. of images to process). Note that lower vertical values represent less mobile energy usage that is desired.

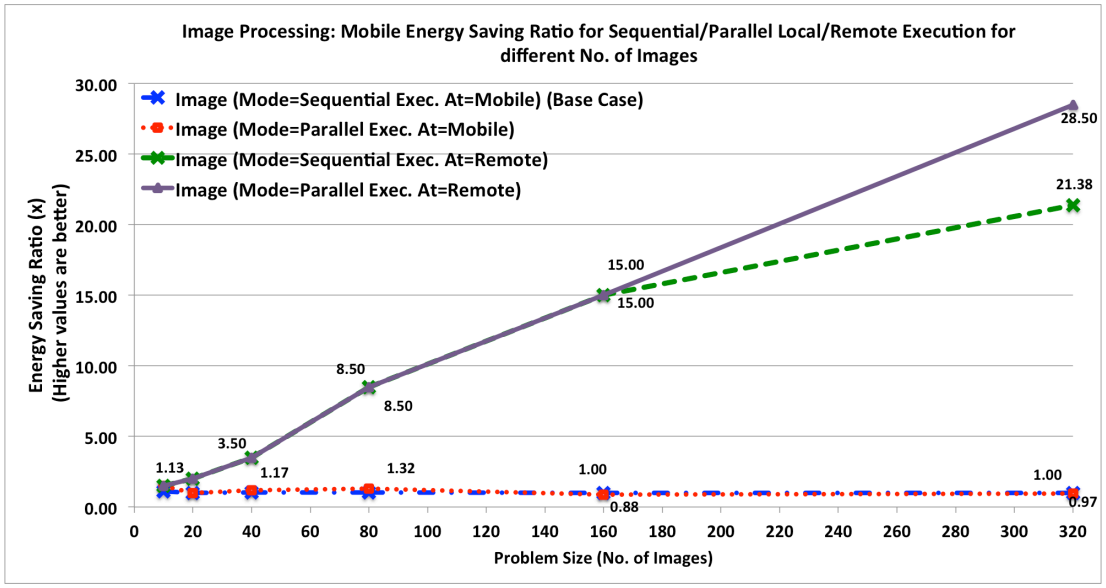


Figure 5.2: Energy saving ratio with different amount of work. Mobile energy saving ratio for local and remote execution of Image Processing application for different amount of work (different no. of images to process). Note that higher vertical values represent more mobile energy saving that is desired.

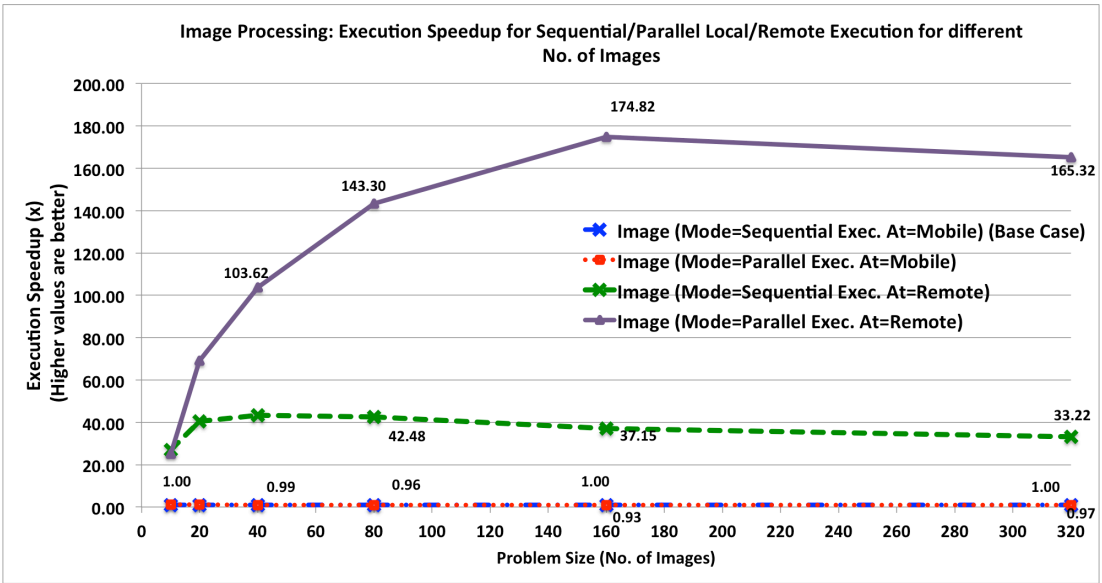


Figure 5.3: Speedup for different amount of work. Execution speedup for local and remote execution of Image Processing application for different amount of work (different no. of images to process). Note that this speedup is a side-effect of offloading for maximizing energy saving on mobile device. Also, note that higher vertical values represent more execution speedup that is desired.

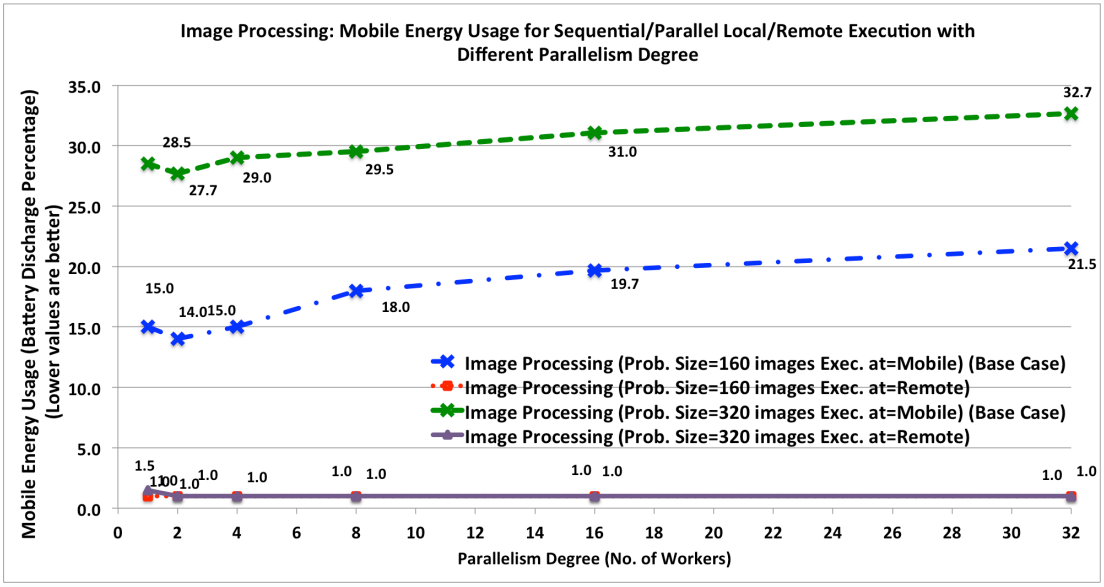


Figure 5.4: Energy usage for local and remote execution. Mobile energy usage for local and remote execution of Image Processing application with different degree of parallelism. Note that lower vertical values represent less mobile energy usage that is desired.

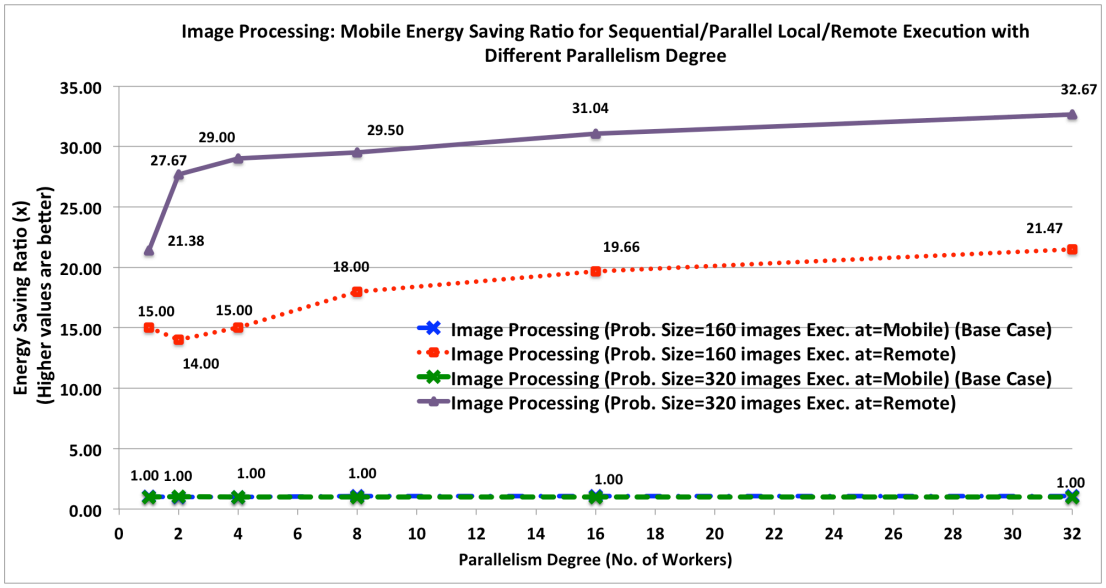


Figure 5.5: Energy saving ratio for local and remote execution. Mobile energy saving ratio for local and remote execution of Image Processing application with different degree of parallelism. Note that higher vertical values represent more mobile energy saving that is desired.

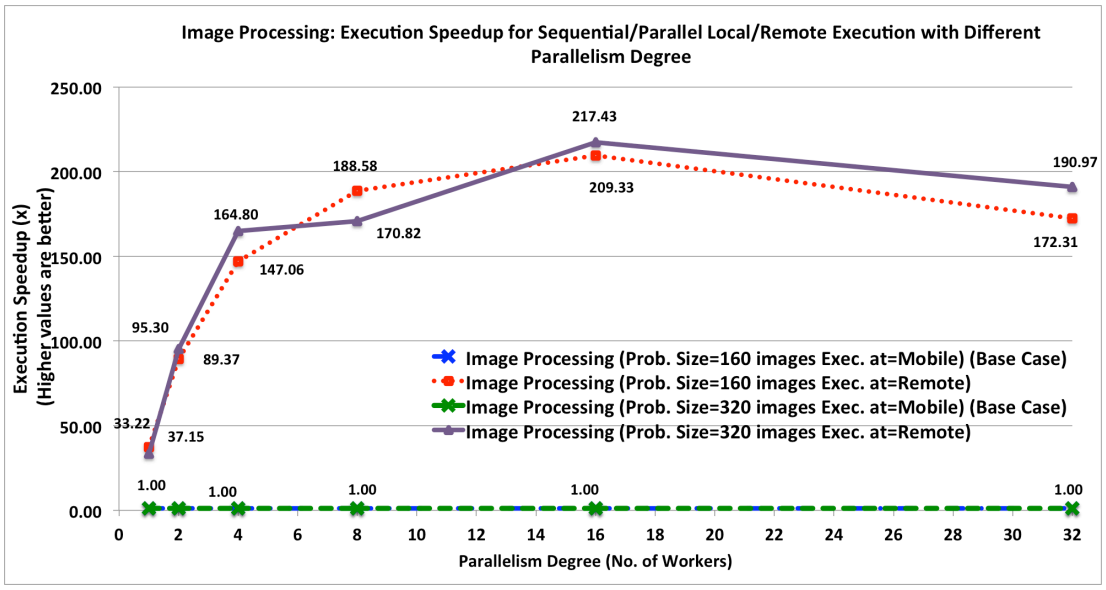


Figure 5.6: Parallelism and speedup for local and remote execution. Execution speedup for local and remote execution of Image Processing application with different degree of parallelism. Note that this speedup is a side-effect of offloading for maximizing energy saving on mobile device. Also, note that higher vertical values represent more execution speedup that is desired.

CHAPTER 6

MONITORING APPLICATION COMPONENT ENERGY CONSUMPTION

In this chapter we discuss our monitoring system. The offloading decision model developed in Chapter 5 shows the need for fine-grained component based energy consumption of mobile applications. While many of the required run-time usage parameters can be directly collected or measured using underlying operating system, e.g. using *procfs/systemfs* or different existing sensors on the mobile device, most of mobile devices do not provide any tool for direct measurement of existing or consumed energy. Even if total energy consumed on the mobile device can be measured, multiple applications are running on the phone at different times and distribution of energy among different applications is not known. Different applications consume significantly different amount of energy and total energy cannot simply be equally distributed among them. This requires development of an energy estimation model that uses different mobile hardware component usage to estimate total energy consumed on the phone. Even when energy consumption of individual applications is known, distribution of that energy among different components of the application remains as a challenge.

Resource limitations of mobile devices imply that any monitoring system should be light weight and energy efficient. The proposed monitoring system for application component energy consumption compromises of tow parts. The first part estimates energy consumption per applications and the second part provides fine-grained energy consumption per application components.

6.1 Energy Consumption of Applications

The accuracy of application-level energy consumption relies on energy estimation model using hardware usage. In this section, we first explain our design of energy estimation model and then clarify how such a model can be calibrated for different mobile devices without using any external measurement tool.

6.1.1 Development of the Energy Estimation Model

While many previous energy models have been developed as a black box relying on assumed relationship between processor energy consumption and other hardware energy consumption, our energy estimation model considers different hardware components independently and predicts total consumed energy based on information of different component states and their energy consumption. It has already been shown that maximum error resulting from considering independence for individual component energy consumption results in less than 6 percent error [44]. As a result, a sum of independent component-specific energy estimates is sufficient to estimate system energy consumption. Using a linear energy model, energy consumption of each application can be defined as Equation 6.1 [52]:

$$E_j = \sum_{i=1}^{M} (\ \beta_i \ * \ u_i^j \ * \ t_i^j \ ) \tag{6.1}$$

where E_j is the energy consumption of application j, β_i is the power coefficient of hardware component i, u_i^j is the hardware component i usage by application j, and t_i^j is the active usage duration of hardware component i by application j. For every application j, the equation goes over all different hardware components of the mobile device and calculates the energy consumption on that hardware component for that application. Assuming that energy consumption of different hardware components are independent and that mobile device has M different components, E_j shows the total energy consumption for a specific application. Note that the accuracy of E_j is affected by correct measurement of β_i, u_i^j, and t_i^j [52]. t_i^j can directly be measured using system clock and processor utilization per applications. u_i^j can also be directly measured using Android operating system profiled data. The only remaining part is β_i that varies significantly for different mobile devices. β_i is usually measured using external power measurement tools in laboratory and is part of the energy model generation process. We will discuss methods to generate β_i without using any external device in the next section. for this section, we assume that β_i coefficients are known.

Knowing energy consumption of different applications allows estimating total mobile energy consumption, Equation 6.2.

$$E = \sum_{j=1}^{N} E_j + (\ P_{base} \ * \ D \ ) \ + \epsilon \tag{6.2}$$

where E is the total energy consumption of the mobile device, E_j is the energy consumption of application j, P_{base} is the base power consumption on the mobile device, D is the device's power-up duration, and ϵ is the noise energy that cannot be estimated from the model. It is

assumed that N different applications are running on the mobile device during time interval D.

Among all different hardware components available on a mobile device, only a few of them consume most of the total energy [44, 52]. In our model, we considered processor, screen, WiFi, cellular, GPS, audio hardware components as the main sources of energy consumption on the mobile device. Using energy models proposed by [42] [44], [52], [107], [108] and [53], we developed an energy estimation model for mobile devices summarized in Table 6.1.

As mentioned before, our energy estimation model includes energy consumption from CPU, screen, GPS, audio, Wifi, and cellular modules. Coefficients for different modules capture relationship between each component state and power consumption of the hardware component. These required coefficients are typically calculated using an external power measurement tool and a calibration program that changes component activity state while keeping all other component states constant. Since processor has to be on while other components are working, processor should be the first component to study. Knowing the processor energy consumption in different states allows excluding energy consumption of CPU from other hardware components when studying different mobile components. Processor energy consumption of an application depends on processor frequency and utilization. Processor parameters can be changed independently from other components while keeping all other components turned off. Thus, it is relatively straight forward to build the power model for processor using an external power meter. For other hardware components, we measure the total energy consumption and then exclude processor energy consumption using the previous CPU energy model. A good example is WiFi module. WiFi cannot work without processor being turned on. Thus, any measurement of energy will include both processor and WiFi component and we need to monitor the state of both hardware components in order to build an energy model for WiFi module. Previous research in this regard [42, 44, 52, 107, 108, 53] have already found important states for different hardware components. GPS module energy consumption is mostly affected by whether GPS module is on or off. Other state factors, such as number of detedcted satellite or received signal strength, are of secondary importance and can be ignored. For WiFi module, the state of the WiFi component is the critical factor. WiFi state can be high power, resulting in higher energy consumption, or low power. The effect of other factors, such as data rate, channel rate, no of packets transferred, are of secondary importance and automatically included in the WiFi state by the operating system. As a result, we only need to include the WiFi state in our power model. Cellular module energy consumption is governed by four main states: off, idle, shared channel, and dedicated channel. In idle state, cellular module only receives paging messages and does not transmit any data. When data needs to be transferred, depending on the rate of data to be

sent, the module will use one shared channel for all applications or one dedicated channel for each application. As a result, the effect of all factors, such as data rate, up-link queue size, down-link queue size, are automatically considered by the operating system in the module's channel state. Audio module power consumption is governed by whether audio component is on or off. The volume has negligible effect on power consumption and is ignored.

6.1.2 Calibration of the Energy Estimation Model

The power models developed in previous section are device dependent and the resulting energy models significantly vary for different types of mobile devices. Even mobile devices having the same processor and screen hardware components can have power variations up to 62% [44, 49]. This requires significant amount of manual work to generate an energy estimation model for each new mobile device and puts creation of energy models behind release of new mobile devices. As a result, we focused on developing a calibration solution that allows building energy estimation models for different mobile devices without requiring any manual work or use of any external power measurement tool.

Mobile devices are capable of showing the remaining amount of battery as a percentage of the total. This is possible by knowing characteristics of Lithium-Ion batteries, that are currently used on all mobile devices. Lithium-Ion batteries are popular because of the their high energy-to-weight ratio, long service lifetimes, and low self-discharge currents [44]. One of the main characteristics of Lithium-Ion batteries is that their voltage drops during discharge and is related to the remaining amount of charge on the battery. Since most mobile devices are already equipped with voltage sensors to be able to use this Lithium-Ion battery characteristic and report the remaining battery charge, we can also use the existing voltage sensor to measure energy consumption on the phone resulting from different activities.

Figure 6.1 shows the trend of remaining battery charge and battery voltage for our experimental mobile device over time while the mobile is put in a constant state using full processing power to perform computations. It can be seen that both measured voltage and reported remaining battery charge are reduced over time, though at different rates.

In order to reveal the relationship between battery voltage and reported remaining battery charge, we can use figure 6.2. It reveals that when remaining battery charge decreases from 100% to 0%, the battery also drops from around 4 volts to 2.85 volts. Knowing this voltage-charge curve allows measuring what percentage of battery capacity used during specific period of time using measured voltage at the beginning and end of the time period.

In order to calculate average power consumption during time interval $[t_1, t_2]$, we can use

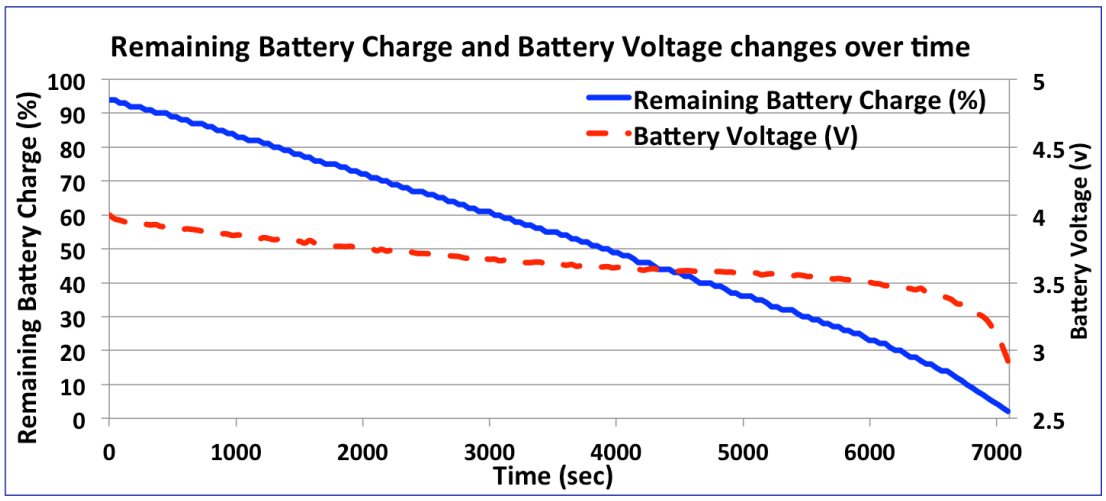


Figure 6.1: Trend of remaining battery charge and battery voltage changes over time. It can be seen that both graphs decrease monotonically over time.

Equation 6.3.

$$P * (t_1 - t_2) = E * (C_{v1} - C_{v2}) \tag{6.3}$$

where P is the average power consumption during time interval $[t_1, t_2]$, E is the total energy capacity of the battery, C_{v1} is remaining battery charge at time t_1, and C_{v2} is remaining battery charge at time t_2. By knowing the percentage of remaining battery at start and end of the time interval, we can calculate what percentage of the total battery energy is used during the time interval.

Knowing average power consumption for different time periods, we use a calibration software to put different mobile hardware components in different states for fixed amount of time and measure the total energy consumption during that time interval. Having the total energy consumption at different states of a specific hardware component, we can calculate the required energy coefficient, β, for that component to build the energy estimation model explained in previous section. Figure 6.3 summarizes different steps required to find the required hardware component energy consumption coefficients and calibrate the energy estimation model. The reason to put mobile device in low power state for 1 minute before each coefficient measurement is to eliminate the impact of the voltage drop across the battery internal resistance [44]. The selected period (15 minutes) is long enough to exceed any potential noise in change of the battery voltage. Previous research have already proved the practicality of this approach and experimental results revealed error less than 1% compared to manual measurement of component hardware energy coefficients [44, 49]. Thus, battery-based automatic power model is as accurate as meter-based power estimation models. Also

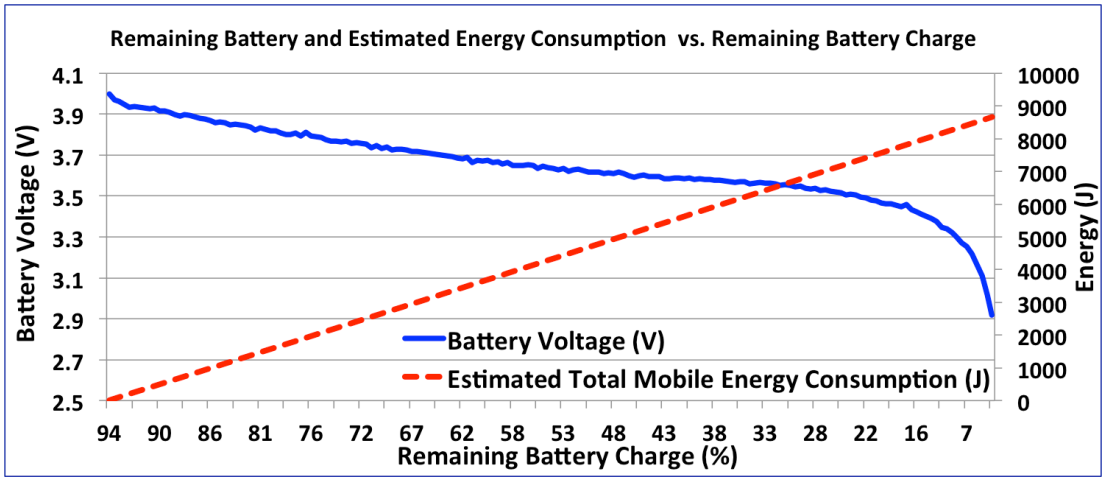


Figure 6.2: Trend of battery voltage and AndroidSalsa consumed Energy changes versus remaining battery charge

evaluation results show that there is 1% error using manual power estimation models for intervals of 10 seconds. Similarly, error resulting from automatic power estimation models is 4% [44, 49]. Thus, automatic power estimation models are accurate enough for our purpose of estimating energy consumption of different applications.

Figure 6.3 shows experimental result for estimating energy consumption of different applications compared to total energy consumption by mobile device. We specifically focus on AndroidSalsa, which provides a SALSA actor environment for running our mobile application. We also profiled the energy consumed by AndroidSalsa applicaiton on communication by considering only components dedicating to communication, WiFi and Cellular modules.

It should be added that both energy capacity of a Lithium-Ion battery and the charge/discharge curve for it can change with discharge current, temperature, and battery age [44, 52, 49]. These factors can potentially affect the accuracy of the generated energy estimation models. In fact, the total energy capacity that we used in our calculations is for a brand new battery. Fortunately for our specific application, the absolute total energy is not important but instead the relative difference of energy values between different applications and different application components is important. Moreover, the change of total energy in normal range of temperature, 73 to 78 degrees of Fahrenheit, is very limited (less than 4% by one estimate [44]). It is only for extreme temperatures or significant temperature changes that the effect of that on total energy and energy-voltage discharge curve becomes significant. Finally, there are two solutions to generate an energy estimation model automatically: i) run the entire calibration code once completely in order to put different hardware components into different states for 15 minutes and measure the energy consumption and find related

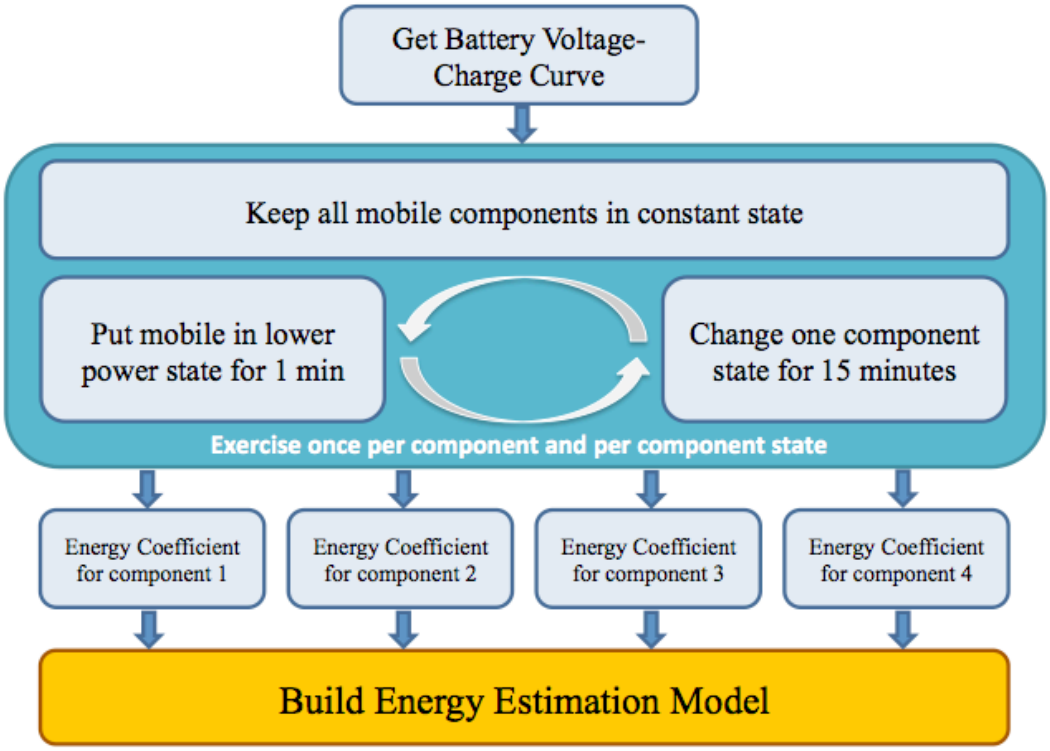


Figure 6.3: Required steps to calibrate the energy estimation model. Steps required to find the required hardware component energy consumption coefficients and calibrate the energy estimation model.

coefficient. This will take a few hours and will consume around one charge of battery. ii) have an online database and gradually collect energy usage of different components at different states from different users using the same phone model. This prevents the initial wait time and consumes less energy for calibration. Fortunately, the energy estimation model only needs to be created once for each mobile device model and any of the two mentioned method can easily be used to generate it.

6.1.3 Experimental Result

In order to evaluate the accuracy of our generated model, we uses an external power meter to monitor instant power of the mobile device. We removed the battery and connected the mobile device to an external source of energy and set the voltage to a fixed value 3.6 volts while allowing current to be changed. Figure 6.5 shows the result from our energy estimation model and the measured values for instant power. It can be seen that error range varies around 10%. Considering the fact that this is the instant power changing rapidly, the result

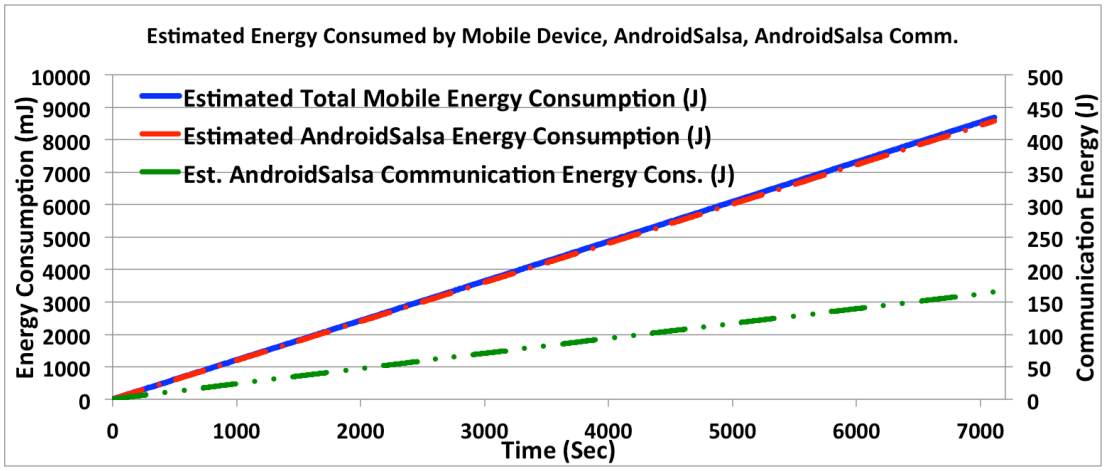


Figure 6.4: Energy profiling per application results. Energy consumption of all running applications on the mobile device, Android Salsa application, and Communication resulting from Android Salsa over time

is within the accuracy that we expect from the model.

Figure 6.6 shows the result for total energy consumption from our energy estimation model and manual measurement. It can be seen from the graph that the two curves follow each other and the resulting average error of 13% is within the acceptable range.

6.2 Fine-grained Energy Consumption of Application Components

In this section we provide a method to attribute aggregate application energy consumption, measured using generated energy consumption model, to individual application components. Let's assume that application under consideration has N different components $C = c_1, c_2, ..., c_N$. Total energy used by a specific application during time interval t_i is E_i and can be represented as:

$$E = \sum_{k=1}^{N} n_{c_k} \ * \ E_{c_k} \tag{6.4}$$

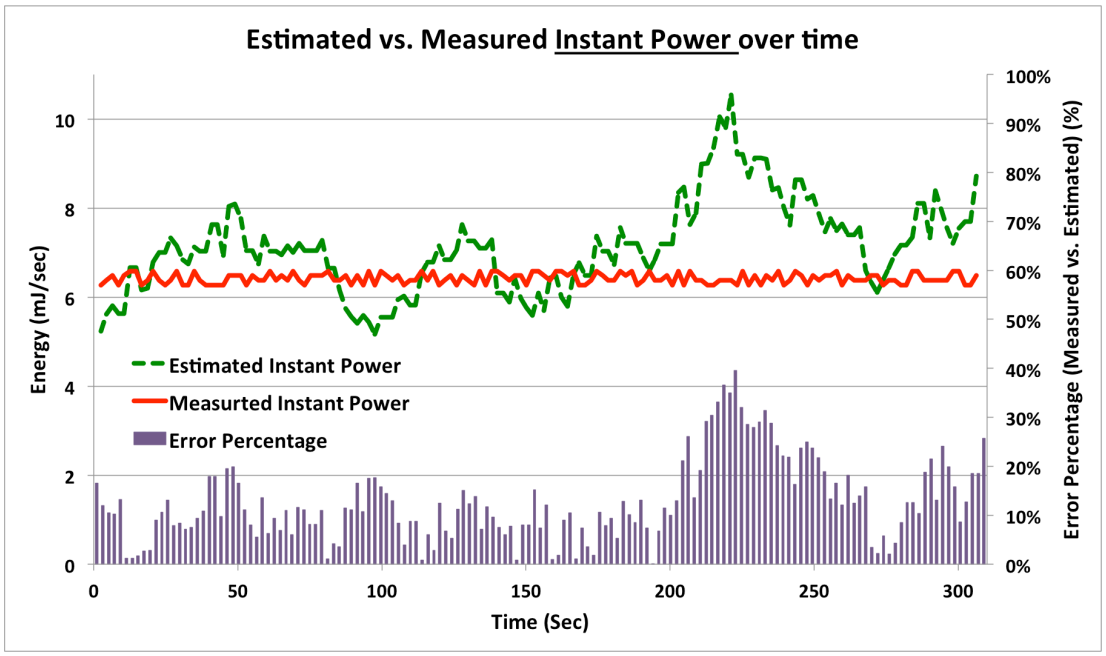


Figure 6.5: Estimated and measured powers. Estimated instant power resulting from the energy consumption model compared to instant power directly measured using external power meter.

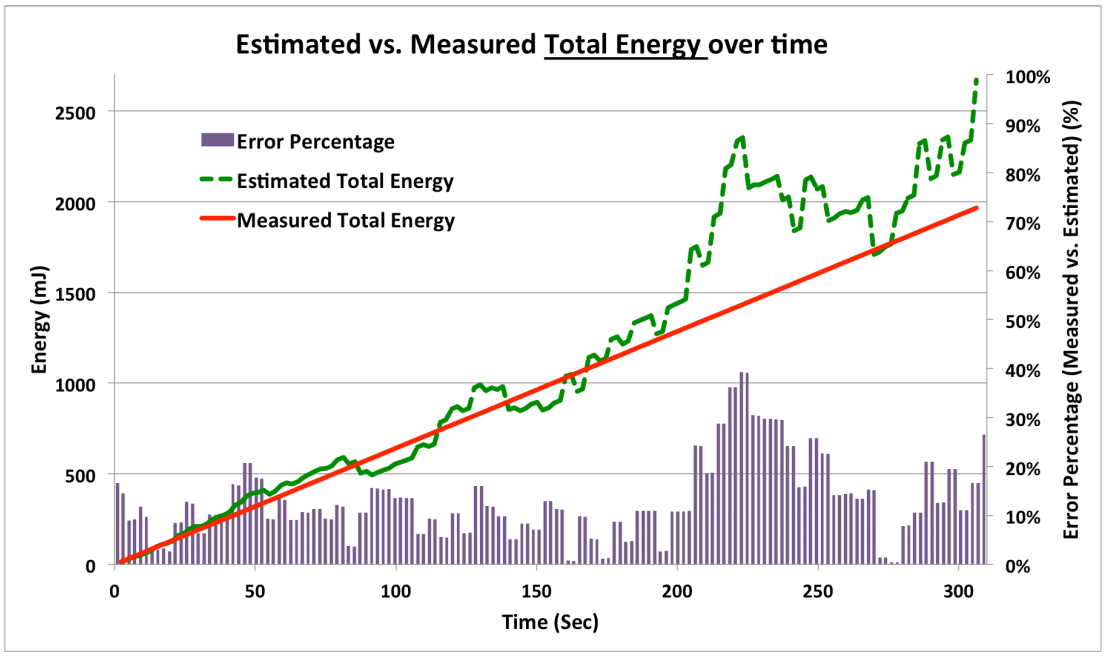


Figure 6.6: Estimated and measured energy. Estimated energy calculated using generated model compared to total energy directly measured using external power meter.

Table 6.1: Power estimation model for different mobile hardware components

Component	Energy Estimation Model
CPU	$P_{CPU} = \beta_{freq}^{CPU} * util + \beta_{freq}^{idle}$
	util: processor utilization, $0 \leq util \leq 100$
	freq: Frequency index, $0 \leq freq \leq n$, n max mode of CPU freq.
	β_{freq}^{CPU}: processor coefficient for different freq.
	β_{freq}^{idle}: processor coefficient for idle state
Screen	$P_{Screen} = \alpha_{LCD} * \beta_{br}^{LCD} * BR_{bg} + (1 - \alpha_{LCD}) * \beta_{br}^{LED} * (\sum BR_{px})/PX$
	$\alpha_{LCD} = 0, 1$, specifies LCD or OLED (AMOLED)
	BR_{bg}: brightness background in LCD
	BR_{px}: brightness current pixel for OLED
	PX: total no. of pixels on screen
	β_{br}^{LCD}: screen coefficient for LCD
	β_{br}^{LED}: screen coefficient for OLED or AMOLED
GPS	$P_{GPS} = \alpha_{GPS} * \beta_{GPS}$
	$\alpha_{GPS} = 0, 1$; specifies whether GPS is on or off
	β_{GPS}: GPS module energy coefficient
WiFi	$P_{WiFi} = \alpha_{WiFi}^{Low} * \beta_{Wifi}^{Low} + \alpha_{WiFi}^{High} * \beta_{Wifi}^{High}$
	$\alpha_{WiFi}^{Low} = 0, 1$, specifies whether Wifi is in low state
	β_{WiFi}^{Low}: WiFi energy coefficient for low state
	$\alpha_{WiFi}^{High} = 0, 1$, specifies whether Wifi is in high state
	β_{WiFi}^{High}: WiFi energy coefficient for high state
Cellular	$P_{Cell} = \alpha_{cell}^{idle} * \beta_{cell}^{idle} + \alpha_{cell}^{shCH} * \beta_{cell}^{shCH} + \alpha_{cell}^{dCH} * \beta_{cell}^{dCH}$
	$\alpha_{cell}^{idle} = 0, 1$, specifies whether Cellular module is in idle state
	β_{cell}^{idle}: cellular module energy coefficient for idle state
	$\alpha_{cell}^{shCH} = 0, 1$, specifies whether Cellular module is in shared channel state
	β_{cell}^{shCH}: cellular module energy coefficient for shared channel state
	$\alpha_{cell}^{dCH} = 0, 1$, specifies whether Cellular module is in dedicated channel state
	β_{cell}^{dCH}: cellular module energy coefficient for dedicated channel state
Audio	$P_{Audio} = \alpha_{Audio} * \beta_{Audio}$
	$\alpha_{Audio} = 0, 1$, specifies whether audio module is on or off
	β_{Audio}: Audio module energy coefficient

CHAPTER 7

FLEXIBLE POLICY-DRIVEN RESTRICTIONS FOR PRIVACY, BUDGET LIMITATION, AND EXECUTION QUALITY OF MOBILE HYBRID CLOUD APPLICATIONS

While having an automatic framework for dynamic offloading of mobile application components to multiple cloud resources is great for maximizing application performance or minimizing mobile energy usage, it cannot be directly applied to all applications. Many organizations, developers, or users benefiting from cloud resources have certain additional requirements, expectations, and policies in terms of how different private or public cloud resources can be used by a mobile application. Without having enough flexibility in the offloading framework to address these additional requirements, many users will not be able to benefit from the cloud resources. In this section, we look at some of these additional requirements, define the required grammar to define policy, and explain how the framework can be customized to address them.

7.1 Defining Privacy for Mobile Hybrid Cloud Applications

One of the primary goals of our proposed framework is to mask the unnecessary details of distribution and move-around of application components from the programmers, while access to different resources and move-around of sensitive or confidential components resources are still restricted based on required policies. This requires the framework to follow authorization rules defined by the organizations, developers, or users. Elastic application components on the cloud should adhere to the property of least privileges. Which permissions a component might have may depend on its execution location, application requirements, or user concerns. Implicit access to device resources may require additional scrutiny when the component is no longer running local to the device [30]. A comprehensive security solution requires authentication, access control, and audition. There has been significant amount of work on authentication and audition for cloud applications in the past and the existing solutions are mature enough to address most applications [30, 109, 110, 111, 112, 112]. While our work primarily focuses on building the transparent connection between mobile device and different cloud resources, privacy of user and applications is of prime importance. As a result,

in this section, we explain our approach for adding policy-based privacy to our framework for restricting the accesses, actions, and move-around of components.

7.1.1 Design of the Authorization System

Our main design goal is to provide a fine-grained authorization system for application components moving between mobile device and different public and private cloud spaces. Using entities defined in Chapter 3, organization, developers and end-users, we want to enable organizations to enforce a organization-wide policy while developers and end-users can fine-tune it. Organization is the primary owner of the data and resources and must be able to keep private and public cloud components separate from each other and define an overall policy in terms of resource usage for different users or different applications. Based on this definition, policy must initially be defined as a high-level organization-wide guideline specifying the restrictions for different applications and users. In addition, specific applications might also need to further tighten these organization-wide policy rules. End-Users or programmers must also be able to further restrict resource usage and component distributions for specific applications. As a result, our framework supports two types of policies: hard policy and soft policy.

Hard policy refers to organization-wide authorization rules defined per user or application by the organization. Users includes different developers inside the organization in addition to external clients. On the other hand, soft policy refers to application-specific authorization rules defined in addition to the organization-wide hard policy. Despite the fact that these two types of policies have complementary roles in increasing system flexibility, soft policy can only tighten the organization-wide policy and not vice versa. In other words, if the organization-wide hard policy allows a specific user or a specific application to access resources A and B, soft policy can only further restrict the access to one of the resources A or B and can never loosen the restrictions by allowing access to a new resource such as C. As a result, hard policies are pre-defined by a central authority, here the organization, and can effectively be used to implement corporate-wide authorization policy for different users or different application. Additionally, soft policies are defined based on the specific needs of the application or user. This perfectly aligns with our initial goal of separating the restriction policy definition from the application logic. Organization defines in advance its hard policy for different applications and users based on its overall goals and characteristic of its cloud architecture. Programmers, working within the organization, later define the application logic without worrying much about compromising the pre-defined organization-wide policy.

The principle of separation of concern is further extended to the developed application by requiring programmers or end-users to specify their arbitrary application-specific restriction rules separate from the program logic and as a soft policy.

Each developed application is used by one end-user. We call each application used by one user as an application instance. Each application instance can have two policy with it: a non-modifiable hard-policy, a modifiable soft-policy. Each application instance initially authenticates itself with a Policy Manager Machine (PMM) and receives a locked unchangeable hard-policy that defines organization-wide authorization rules defined by the organization. Each organization can define its authorization policy as one policy for all users, one policy for all applications, one policy per application, one policy per per user, or one policy per application instance. At the end, each application instance can acquire one locked hard policy from the policy manager machine. In addition, each application instance can have one soft-policy. Developers can define the initial soft-policy per application or per application instance. They can also allow end-users to change all or part of this soft-policy through the application. Thus, end-users can modify soft-policy through application, based on rules defined by the developers. At the end, each application instance can have one soft-policy in addition to the locked hard-policy. We follow XACML usage model [93] and assume a Policy Enforcement Point (PEP) as part of our elasticity manager. PEP is responsible for protecting authorization rules, sending a request containing description of the attempted action to a Policy Decision Point (PDP) for evaluation against available hard and soft policies. The PDP evaluates the requested action and returns an authorization decision for the PEP to enforce.

As was mentioned before, our cloud application consists of distributed components storing data or performing computation and scattering between cloud spaces and end-user device. As a result, our authorization framework needs to be able to apply the restriction rules at the granularity of actors. It still allows defining those rules at higher-level entities, such as groups or sets of actors, but it recursively propagates all those specified authorizations (permissions or denials) to all actors contained within that set at run-time. This makes it easy to specify authorizations holding for a larger set of actors (on the whole system in case ALL is used) and have it propagated to all the actors within that set til stopped by an explicit conflicting restriction rules (Damiani et al. 2002, Jajodia et al. 2001, Lunt 1989). Actor frameworks allow multiple actors to be placed together in a container, called actor-System or theaters, to share common attributes. We respect this structuring in our grammar and allow authorization rules to be defined on actors, actor-systems, set of actors (called Group), set of actor-systems (called Location), or subset of multiple actors and actor-systems (called Selection).

There is a significant distinction between traditional access control systems and our proposed authorization model. While access control models restrict access to different components or resources, our mobile hybrid cloud framework provides more than access restriction. Actor programming paradigm allows an actor to send and receive messages, create new actors, or migrate to new locations. As a result, our authorization grammar must allow defining rules regulating all these actions. Note that these actions are usually bidirectional, meaning that if actor 1 is allowed to send to actor 2, then actor 2 must also be allowed to receive from actor 1 in order for the policy to be consistent. If any of these two actions are not explicitly allowed as part of the policy, the framework automatically rejects both actions, as they will always happen together.

7.1.2 Mobile Hybrid Cloud Authorization Grammar

Authorization decisions are made based on the attributes of the requester, the resource, and the requested action using policy-defined rules. As a result, definition of our authorization policy is composed of defining the authorization entities and their required attributes in addition to defining rules and desired rule-orderings. In this section we will look at the required grammar to define these. Note that any new definition of a grammar is surrounded by curly braces "{ }", sequence of elements are separated by comma ",", optional entities are surrounded by the brackets "[ ]", and choices are separated by vertical bar " — ". Moreover, every definition in our grammar must have a unique name working as its identifier throughout the entire policy.

Authorization-Related Entities

As mentioned earlier, our cloud application model consists of components called actors storing data or computational code. Based on this view, actors are the smallest entities in our programming model and thus, the finest granularity on which we can define access restriction. In order to provide location transparency, multiple actors running on one instance of JVM on one machine are placed inside a container, called actor-system, or Theater as in SALSA. Actor-systems main task is to mask the underlying details of calling remote actors, migrating actors to remote locations, enforcing authorization rules, and dynamic management of actor distribution. Our grammar supports both actors and actor-systems and Table 7.1 below shows the definition of these entities.

Every actor is defined by its related reference, logical path to reach the element at runtime environment, in addition to its containing actor-system. Authorization framework uses

Table 7.1: Definition of authorization-related entities

Entity	**Definition**
Actor	{Name, Static (Reference, ActorSystem) | Dynamic ([Reference], [ActorSystem]), Quality([Quality-Level] | [(ActorSystem,Quality-Level)]) }
ActorSystem	{Name, Static (URL, Port) | Dynamic ([URL], [Port]), Budget-Type (UNLIMITED | FIXED | RATE), Budget}
AnonymousActors	{Name, Ref-ActorSystem | ALL, Existence (ALLOWED | FORBIDDEN), Limitation (LIMITED | UNLIMITED), max-number}
AnonymousActorSystems	{Name, URL | ANONYMOUS-URLs, Creation (ALLOWED | FORBIDDEN), ActorSystem-Limitation (LIMITED | UNLIMITED), max-number-of-ActorSystems, Actors-per-ActorSystem-Limitation(LIMITED|UNLIMITED), Max-Number-of-Actors-per-ActorSystem}

these attributes to bind actors defined in the policy to real-world application components. Both static and dynamic bindings are supported with static type using fixed known actor reference and actor-system to bind an actor definition to its related run-time component, while dynamic type acting as a placeholder for an actor not fully known at the time of writing policy file. Note that actor-system property of an actor definition uses the name of an already-defined actor-system as an ID to reach it.

Table 7.1 also shows rule for defining an actor-system. In our grammar, every actor belongs to an actor-system. Actor-system is defined by specifying its related URL/IP address and the listening port number. Since more than one actor-system can run in one JVM on a specific machine, both URL and port numbers are needed to connect to different actor-systems running on the same machine. Most actor frameworks allows using actor-systems URL and port number to interact with remote actors. However, they usually provide communication optimization for local communication between actors running on the same JVM. While actors are extremely lightweight and many of them can efficiently be executed within one actor-system on a normal machine, actor-systems are usually not as lightweight as actors. Actor-systems usually need to provide a thread-pool for the use of actors, perform bookkeeping operation for messages being transmitted between actors, and properly schedule the actors. Fortunately, the use of actor-system hides all these underlying details from the programmer or the authorization policy writer.

Explicit actor and actor-system definitions can be used to define entities for application or resources with known architecture. A good example is defining a specific actor as the gateway to a public cloud space. Since organizations know the architecture of their cloud spaces, specific actors from these spaces can be selected, defined as an entity, and later used by rules defining the required gateway access restrictions. On the other hand, details of the to-be-developed applications are unknown at the time of writing organization-wide policy. In order for our grammar to be able to control the existence and activities of these to-be-developed application-specific components, anonymous types of entities are defined as part of the proposed grammar. *Anonymous-actor* rule allows restricting the creation and number of unknown actors in a reference-actor-system. Similarly, *anonymous-actor-system* rule allows controlling the creation and the number of unknown actor-systems, in addition to potential unknown actors to be placed in them. Using specific URL/IP restricts the actor-system creation on a specific machine while using anonymous-URL generalizes the definition to any unknown machines. Moreover, limitation can also be placed on the number of such actor-systems, in addition to the number of actors to be created on them.

7.1.3 Grouping, Selection, and Binding

Although definitions of Table 7.1 can be used to define individual actors and actor-systems, in many cases it is easier to group several entities and treat them as one. Table 7.2 presents required grammar to support grouping. Group definition puts several actors together into one virtual container and allows placing both known actors and unknown anonymous-actors together into one group. Location provides the same grouping functionality but for actor-systems. One or several previously defined actor-systems, locations or even unknown anonymous-actor-systems can be placed into one container location entity.

Instead of specifying individual entities to form a container, selection definition can be used to pick entities based on a condition. It starts with the initial set defined by the from field and removes entities having a specific attribute condition defined by using attribute names, types, values and operators.

Previously defined sets can be combined and modified using the set-operation definition. This modification includes adding/removing object entities to/from subject set in addition to intersecting/combining subject and object sets. Set-operation evaluation is performed at the run-time and its result depends on existing bound run-time entities. As an example, the result of executing a set-operation requesting to add all anonymous-actors to group1 will depend on the run-time actors defined by the application and belonging to the anonymous-

Table 7.2: Definition of authorization entity grouping, selection, and binding

Entity	**Definition**
Group	{Name, Actors | Groups | AnonymousActors | ALL (Actors | AnonymousActors)}
Location	{Name, ActorSystems | Locations | AnonymousActorSystems | ALL (ActorSystems | AnonymousActorSystems)}
Selection	{Name, From (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations), Condition (Attribute-Name, Attribute-Type, Condition-Operation (== | != | ≥ | >| ≤ | <| HAVE | NOT-HAVE | CONTAIN | NOT-CONTAIN), Attribute-Value) }
Set-Operation	{Name, Subject (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | ALL), Object (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | ALL), Operator (UNION | INTERSECTION | ADD | REMOVE) }
Assignment	{Name, Actors (Reference-Actor-Name, Actor-Reference-Value, ActorSystem-Value) | ActorSystems (Reference-ActorSystem-Name, ActorSystem-URL-Value, ActorSystem-Port-Value) }

actors entity.

In order to bind previous dynamic actors and actor-systems to specific run-time component, assignment definition can be used. Any remaining unbound dynamic actor or actor-system is in passive state and will be ignored while enforcing the policy. Assignment definition can then be used to bind them to specific actors or actor-systems and change their passive state to active at any time.

7.1.4 Policy Description

The main goal of writing a policy file is to define required authorization rules on actions among distributed application components, known as actors. Previous defined grammar allows defining entities and grouping or selecting them that is a pre-requisite for defining restriction rules. Table 7.3 shows the required defining grammar for expressing authorization rules and specifying their enforcement ordering.

Each rule definition regulates one action from subject entities to be performed on object entities. Actions include all allowable actions within an actor framework: sending, receiving,

Table 7.3: Definition of authorization rules and their enforcement ordering

Entity	**Definition**
Rule	{ Name, Subject (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | ALL), Object (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | ALL), Actions (SEND-TO | RECEIVE-FROM | MIGRATE-TO | BE-MIGRATED-FROM | CREATE-AT | BE-CREATED-AT-BY | ALL), Permission (ALLOWED | DISALLOWED) }
Rule-Order	{ Name, Subject (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | Rules | ALL), Object (Actors | ActorSystems | AnonymousActors | AnonymousActorSystems | Groups | Locations | Selections | Rules | ALL), Order (PRECEDENCE | SUBSEQUENCE) }

migrating, and creating. This allows regulating actions, move-around, and communication between actor components of a mobile hybrid cloud application.

Since defined rules are not all equally important, *rule-order* definition allows prioritizing specific rules to give them precedence or sub-sequence order over one or several other rules. This highly affects the authorization evaluation result and is one of the most important issues of defining a comprehensive policy.

7.1.5 Policy Evaluation

In a mobile hybrid cloud framework with authorization restrictions, every requested action by the subject has to be approved by the authorization framework before being performed on the object. To make a decision, authorization system has to evaluate the defined policy rules. However, it is possible for different policy rules to contradict each other, as rules are human-defined by different parties, organization and developers, at different times, at different levels, and for different purposes. Moreover, our framework supports both positive (permissions) and negative (denials) access rules. The reason for having both positive and negative authorizations is to provide a simple and effective way to authorization rules applicable to different sets and groups of actor components with supports for exceptions. Having both permission and denial rules further complicates the development of a comprehensive conflict resolution strategy. As mentioned earlier, our framework prioritizes hard

policy rules, defined at a higher level by the organization, over soft policy rules, defined by programmers for individual applications or instances. Prioritizing hard policy restriction rules over soft policy rules allows resolving any potential conflict between hard and soft policies. However, it is still possible for one type of policy to internally have contradicting rules. Although our grammar allows policy makers to explicitly prioritize some of the action control rules over others within a specific policy, our conflict resolution policy is designed to always prioritize action denials over permissions in case of any remaining ambiguity between two contradicting rules. Furthermore, our framework follows a closed action control system model by rejecting any request from a user without an explicit policy rule permitting the action. In other words, in case of no explicit rules allowing an action, the framework simply denies it.

Every authorization rule can be summarized as a five-tuple of the form <Subject, Object, Action, Sign, Type >. Subject and object are the entities between which the specific action is being restricted. *Sign* can be allowance (+) or prohibition (-) and *Type* covers hard policy (H) or Soft policy (S). In order to decide on any requested action, the authorization system has to process rules in a meaningful way from the most prioritized one, usually the most specific rule, to the least prioritized one, the most general one. Based on our conflict resolution policy of prioritizing denial over contradicting allowance, the rule processing sequence can be ordered as below:

1. Hard explicit prohibition rules
2. Soft explicit prohibition rules
3. Hard explicit allowance rules
4. Soft explicit allowance rules
5. No explicit rules defined

For every requested action, the rules have to be processed in this order and the decision can be made as soon as a matching rule is found. However, as can be seen, this does not include the user explicit requested rule ordering. The revised ordering sequence based on user specific rule-orderings can be seen below:

1. Unordered hard explicit prohibition rules
2. Unordered soft explicit prohibition rules
3. Ordered hard explicit allowance/prohibition rules

4. Ordered soft explicit allowance/prohibition rules

5. Ordered/unordered hard explicit allowance rules

6. Ordered/unordered soft explicit allowance rules

7. No explicit rules defined

In order to evaluate the rules in the sequence defined above, we can stop at the first governing rule when processing categories 1, 2, 5, and 6. In other words, if there is any unordered explicit prohibition rule (categories 1 and 2) or no explicit governing rule (category 7), we simply reject the action. If there is any ordered/unordered explicit allowance rule, the action is accepted (categories 5 and 6). However, for ordered allowance and prohibition rules, we cannot stop at the first governing rule and we need to process them all before accepting or rejecting an action. This category of rules (3 and 4) forms a directed graph where nodes are the subjects and objects of the rule definitions connected by directed edges showing the direction of the defined priority, precedence or subsequence. In order to evaluate them, we need to do a level-order traversal on the graph. Obviously, the graph needs to be a directed acyclic graph (DAG) in order to prevent any inconsistency between governing rules. As a result, our framework first checks the graph and notifies the user in case of any cycle. However, in order to make a decision, not all the levels of the graph have to be traversed. A decision can be made as soon as one level is completely traversed and at least one governing rule is found. If no governing rule is found, then the next level has to be completely traversed. Decision can be made at the end of complete traversing of any level, if at least one governing rule is found. Algorithm shows the pseudo-code of rule evaluation algorithm for different categories of rules:

7.1.6 Implementation of the Policy-based Authorization System

In order to simplify the process of defining authorization policy and transferring it between different platforms, policies can be defined as XML documents, that will be validated according to a designed schema. Figure 7.1, Figure 7.2, Figure 7.3, and Figure 7.4 show parts of such validation schema tree and the related XML Schema Design (XSD) file. Both organization and application policies are presented as labeled tree containing a node for each element, attribute, and value associated with fixed attributes. There is an arc between an element and an element/attribute belonging to it, labeled with the cardinality of the relationship, and between a fixed attribute and each of its value(s). Note that since elements and

Algorithm Pseudo-code for rule evaluation algorithm

```
bool Policy_Eval(Subject s, Object o, Action a) {
for all rulesr ∈ categoriesa&b do
  if r.action.contains(a) &&
  r.subject.contains(s) &&
  r.object.contains(o) then
    //note: an action has to be disallowance to be placed in categories a & b
    return false
  end if
end for
for all categoriesi ∈ categoriesc&d do
  for all leveljoflevel_order_traversal(graph(categories(i).rules)) do
    bool matching_rule_found = false;
    bool matching_rule_result = false;
    for all rulesrdefinedatlevelj do
      if r.action.contains(a) &&
      r.subject.contains(s) &&
      r.object.contains(o) then
        if !matching_rule_found && r.permission.equalsIgnoreCase("Allowance") then
          matching_rule_result = true
        end if
        matching_rule_result = true
      end if
    end for
    if matching_rule_found then
      return matching_rule_result;
    end if
  end for
end for
for all rulesr ∈ categoriese&f do
  if r.action.contains(a) &&
  r.subject.contains(s) &&
  r.object.contains(o) then
    //note: an action has to be allowance to be placed in categories e & f
    return true
  end if
end for
//closed authorization system rejects any action not explicitly allowed:
return false;
```

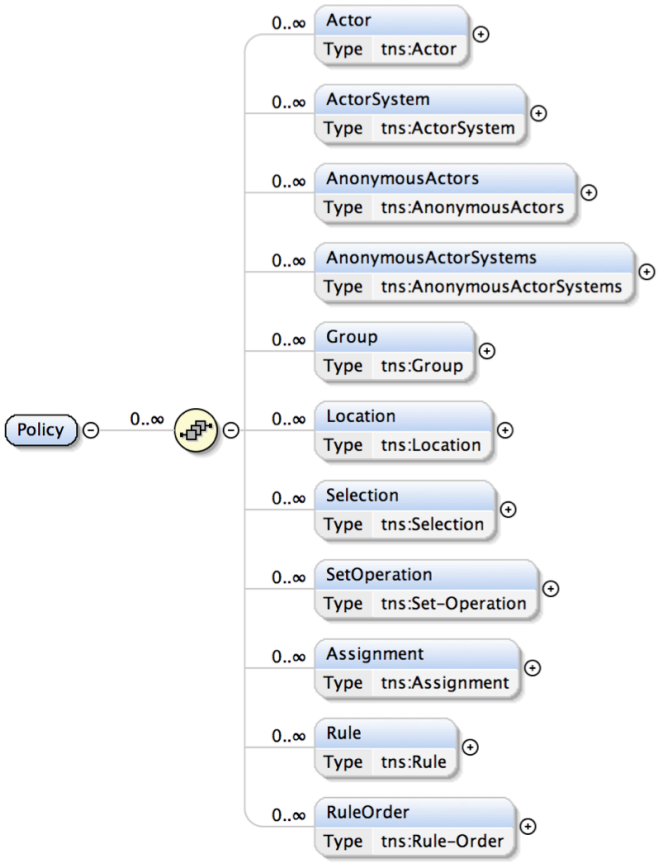


Figure 7.1: Policy schema tree

attributes defined in a policy file may appear in an XML document zero (optional elements), one, or multiple times, according to their cardinality constraints, the structure shown in the Figure 7.2 is not rigid. Different authorization policy files may differ in the number and structure of elements.

All our grammar definitions are further translated to XML schema design format and stored in an XSD file that is used to validate any user-defined policy file before use. Figure 2.a and 2.b below shows a sample definition of actor entity grammar in the tree and xsd format. Other definitions are also defined in a similar way but more complex and are skipped here to save space.

7.1.7 An Example

This section presents how the proposed authorization grammar can be used to construct required policies for one of our benchmark applications, *Image* or image processing application. Details of the implementation of the *Image* problem was explained in Chapter 3.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.shiftehfar.org/sCloud/sCloudPolicy"
    xmlns:tns="http://www.shiftehfar.org/sCloud/sCloudPolicy" elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <xs:element name="Policy">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded" minOccurs="0">
                <xs:element name="Actor" type="tns:Actor" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="ActorSystem" type="tns:ActorSystem" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xs:element name="AnonymousActors" type="tns:AnonymousActors" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xs:element name="AnonymousActorSystems" type="tns:AnonymousActorSystems"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Group" type="tns:Group" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Location" type="tns:Location" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Selection" type="tns:Selection" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xs:element name="SetOperation" type="tns:Set-Operation" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xs:element name="Assignment" type="tns:Assignment" minOccurs="0"
                    maxOccurs="unbounded"/>
                <xs:element name="Rule" type="tns:Rule" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="RuleOrder" type="tns:Rule-Order" minOccurs="0"
                    maxOccurs="unbounded"/>

            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Figure 7.2: Part of policy schema definition as XML schema design file (xsd)

Figure 7.5 shows a summary of different modules involved in *Image* application.

In order to respect the privacy of the clients and prevent potential vulnerabilities of storing private data on third-party machines, the owner organization breaks down the database of all known faces into two main categories: i) faces that are publicly known and contains no sensitive or confidential information and can be stored on public cloud space. ii) faces that contains sensitive or confidential information and should never be placed on third party machines even if encrypted. In order to make this more interesting, different subcategories are also considered for each main category. Let's assume that the goal is recognize people entering the country at an airport through image processing of photos taken by security officers at the airport. Based on this scenario, the two main categories for database of known faces include known criminals whose faces are publicly announced and normal people and citizens whose face should not be stored on any third party machines. Furthermore, faces of known criminals are divided into international criminals and national criminals. Similarly, faces of normal people are divided into citizens, permanent residents and people with visa. Figure 7.6 shows these databases and their storage on different public and private cloud spaces.

As can be seen in Figure 7.6, DB1 stores faces of citizens, DB2 stores faces of permanent residents, and DB3 stores faces of people with visa. Similarly, DB4 stores faces of international criminals and DB5 stores faces of national criminals. Image processing application

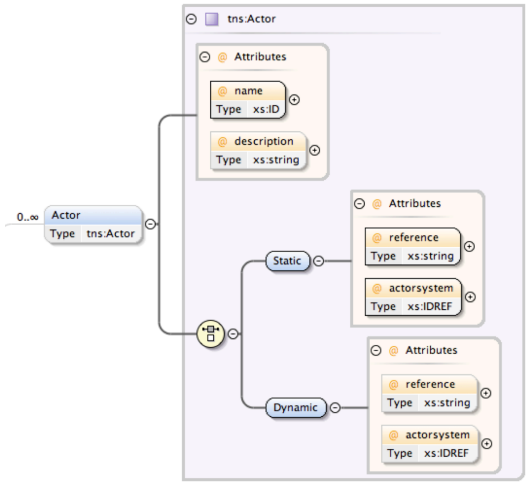


Figure 7.3: Actor entity attributes tree

allows comparing a picture taken by phone with these databases and recognize the people in the photo. The main design goal of developing an authorization policy for this application is to keep private confidential data, normal people databases, separate from unauthorized users and provide different levels of restrictions for different types of users ranging from security officers, who has access to all databases, to normal airport staff, who should only be able to recognize criminals.

Authorization Policy Development Process

In the following, we briefly present how to use the proposed grammar in building the required authorization policy instance for image processing application. In addition to the authorization goal of restricting access of different users to the private and public cloud spaces, a service oriented architecture that allows access to cloud resources through gateway service providers is considered. Private and public cloud spaces are structured as a bunch of services with gateway actors responding to requests. This architecture allows cloud services to be developed and maintained in advance of application development in addition to keeping cloud resources transparent and separate from the rest of the system. Obviously, this is a customized architecture for this specific organization but shows the flexibility of the

```
<xs:complexType name="Actor">
    <xs:choice maxOccurs="1" minOccurs="1">

        <!-- Choice1: Static Actor Type -->
        <xs:element name="Static" maxOccurs="1" minOccurs="1">
            <xs:complexType>
                <xs:attribute name="reference" type="xs:string" use="required"/>
                <xs:attribute name="actorsystem" type="xs:IDREF" use="required"/>
            </xs:complexType>
        </xs:element>

        <!-- Choice2: Dynamic Actor Type -->
        <xs:element name="Dynamic" maxOccurs="1" minOccurs="1">
            <xs:complexType>
                <xs:attribute name="reference" type="xs:string" use="optional"/>
                <xs:attribute name="actorsystem" type="xs:IDREF" use="optional"/>
            </xs:complexType>
        </xs:element>
    </xs:choice>

    <xs:attribute name="name" type="xs:ID" use="required"/>
    <xs:attribute name="description" type="xs:string" use="optional"/>
</xs:complexType>
```

Figure 7.4: Actor entity attributes definition as XML schema design file (xsd)

proposed grammar and its capability to adjust. Figure 7.7 shows the internal architecture of databases within private and public cloud spaces.

Architecture shown in Figure 7.7 considers one actor component for accessing each database and reading face images, actors 1, 2, 3, 7, and 8. In addition, there are actors processing those face images, extracting features, and generating a pre-processed set of known face features for all the faces in that database, actors 4, 5, 6, 9, 10. Databases stored in private and public cloud spaces are contained within an actor-system, actor-systems 1 and 3. Services offered by each of these actor-systems can be accessed through a gateway actor, privateGateway actor and publicGateway actor. Since we are considering offloading, each of private and public cloud spaces provide additional actor-system for application components to be offloaded, actor-system 2 and actor-system 4.

As we mentioned earlier, we want to restrict accesses to public and private cloud databases based on the users of our application. These cloud spaces are owned by the organization and the governing authorization policies are defined by the organization and as a hard policy. In order to build the required policy, let's start by the private cloud space that stores three of the confidential databases and provide access to those services through privateGateway actor. To better manage the private cloud, two actor-systems are created. ActorSystem1 manages confidential databases and ActorSystem2 allows offloading resource for applications. Figure 7.8 shows the detailed structure of the private cloud.

The process of development of the required policy is to ,first, define the required entities, second, define the required rules, and finally, define the required rule-orderings to prioritize more specific rules over more general ones. Table 7.4 shows details of such authorization

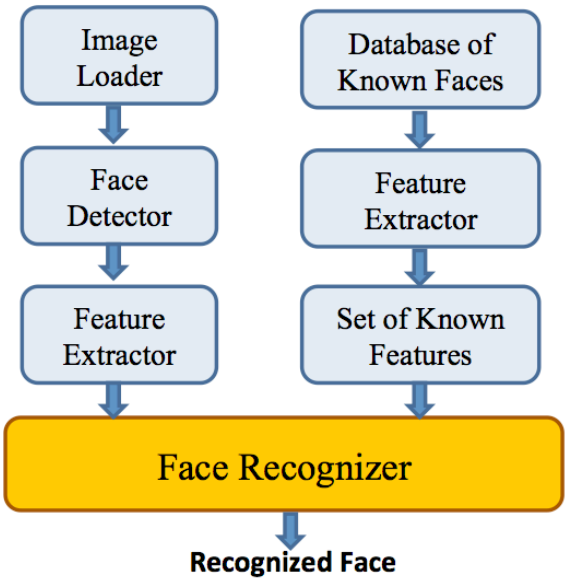


Figure 7.5: Modules involved in image processing application

policy defined for the private cloud and as part of the hard policy file.

As can be seen, the organization enforces a specific architecture in terms of using private cloud databases by defining a gateway actor(ActorPrivateGate in line 2 and 13). This gateway actor receives requests from outside world, accesses databases to perform the actions, and sends back the result. In this specific example, private cloud actor-system provides 3 different services, to access set of extracted features from DB1, DB2, and DB3. Details of actors proving these services can be developed independently from the specific *Image* application using them. These service actors are statically defined, in lines 4,5,6,7,8. In addition, creation of additional actors within this this actor-system and creation of any other actor-system on private cloud space are prohibited, lines 9 and 10. Actor-system2 is defined to allow offloading components from specific applications, line 16. As a result, this actor-system allows creation of dynamic actors based on application requirements, line 17.

Having the required private cloud entities defined, the next step is to define required rules restricting their interactions with the rest of the system. For ActorSystem1, gateway actor, privateGateway, is the only entity that should be accessed by external components. This is reached by first isolating all the actors in the actor-system1 from communicating to any other entity (lines 11), then defining overriding rules to allow their communication within their own actor-system (lines 12), and finally allowing the gateway actor to freely communicate with all other entities (lines 13). Note that after defining the required rules, the required ordering must also be specified (lines 14, and 15).

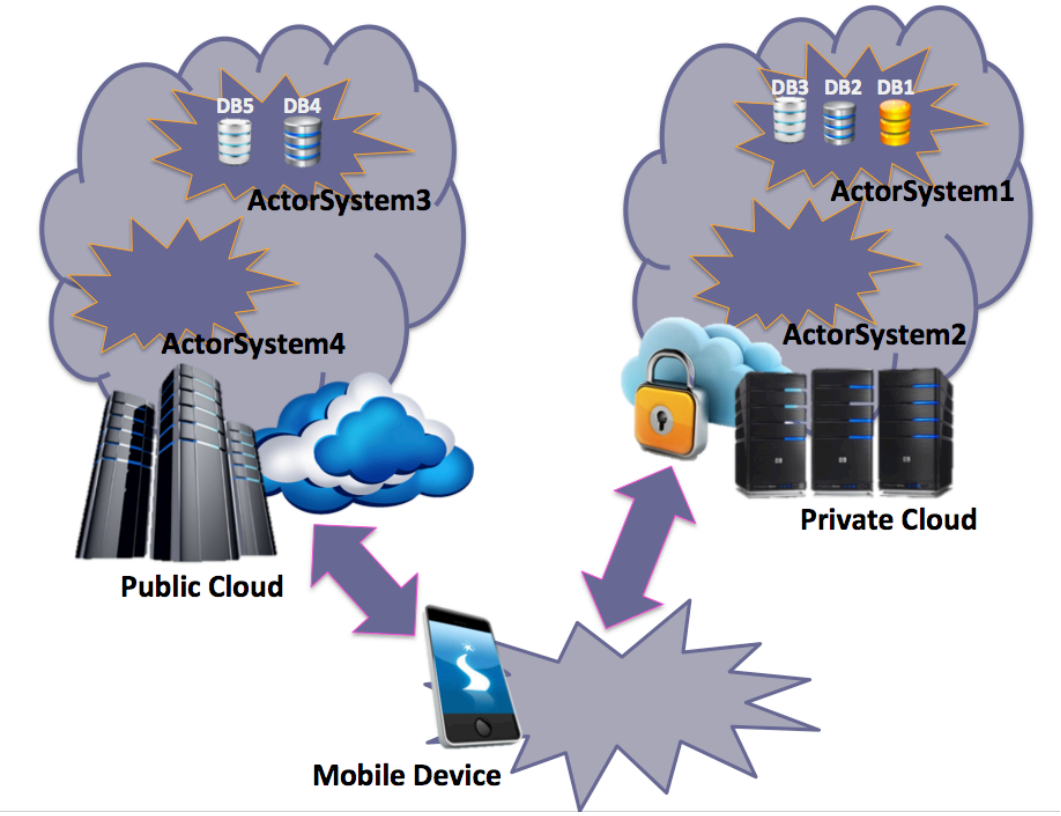


Figure 7.6: Modules involved in image processing application

The structure of the public cloud is very similar to the private cloud with the exception that only two databases are stored there. Figure 7.9 shows the detailed structure of the private cloud. ActorSystem3 consists of a gateway Actor, PublicGateway actor, and two service actors to access databases and provide extracted feature sets, Actor9 and Actor 10.

The steps needed for the development of the required policy for a public cloud are similar to those needed for the development of the required policy in a private cloud and is defined in the same organization-wide hard policy. Table 7.5 shows the required policy file for the public cloud space:

Having organization-wide public and private cloud policies defined, the only remaining step for the organization-wide hard policy is to allow different applications to define their own structure and have access to these cloud resources. Note that hard policy is defined in advance of any application development and is not bound to any specific application architecture. Instead, it defines restrictions for existing services based on their pre-defined architecture. Allowing user applications to create any arbitrary architecture is relatively easy and can be done using dynamic binding property of the framework. Table 7.6 shows the required policy file for the end-user applications:

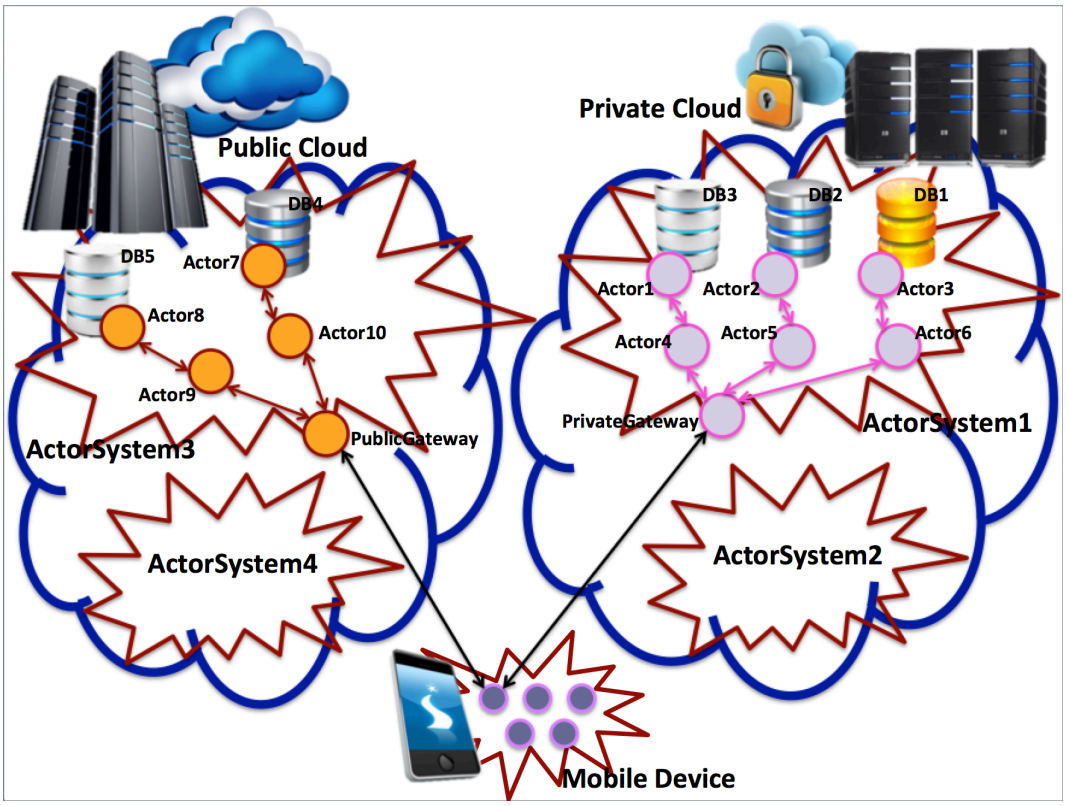


Figure 7.7: Internal structure and organization of databases within the private and public cloud spaces

Table 7.4, Table 7.5, and Table 7.6 together form the organization-wide hard policy defined for an application instance that has access to both private and public cloud databases. As a result, the user using this hard-policy can check faces in a photo against all public and confidential databases. While this high level of access can be useful for a security officer at an airport, other staff members or ordinary people using the application must not be able to access the confidential private cloud databases. As a result, the hard-policy needs to be adjusted to provide correct access rights for normal people. This is relatively straight forward using our defined grammar. Since the only change is to revoke access to private cloud databases, the only required change to the previous policy file is to remove rules providing those accesses, lines 13 and 15. Removing these two lines, prevents access to private cloud databases for a normal application user and restrict his access to public cloud databases. Note that the user can still use private cloud for offloading purposes and only access to private cloud databases is removed.

In addition to the hard policy defined by organizations, developers can define additional required soft policy to tighten organization-wide policies for their specific application. Developers can also provide end-users with the ability to adjust these soft policies based on

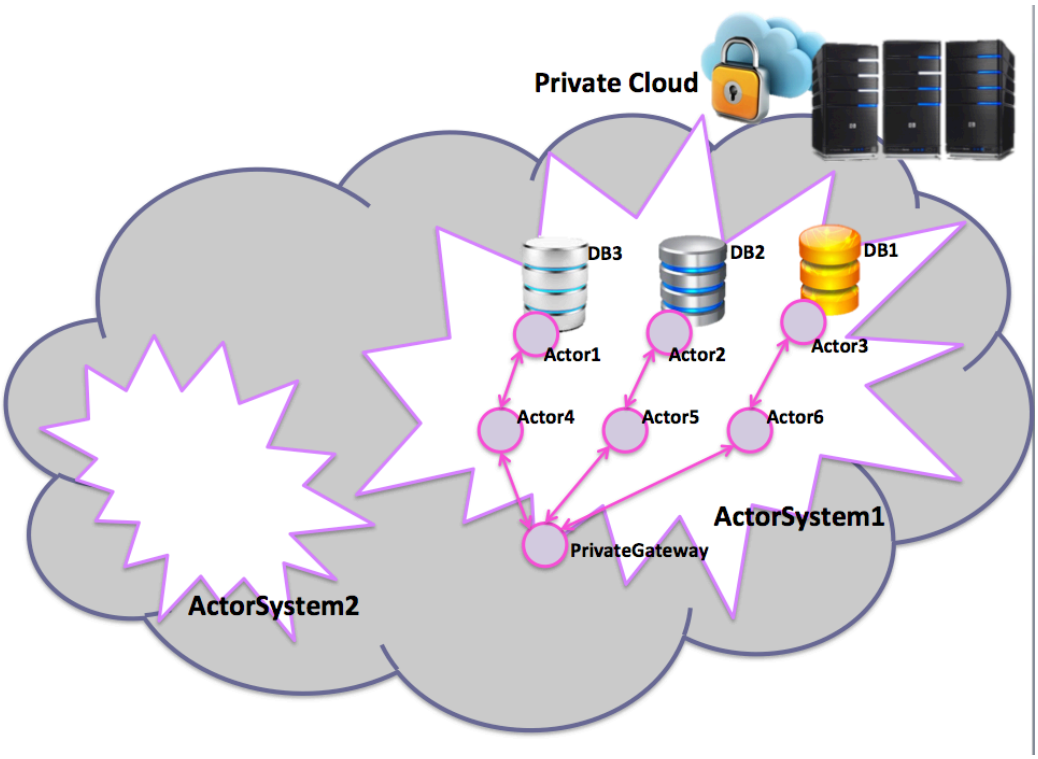


Figure 7.8: The structure of the private cloud space consisting of 2 ActorSystems. ActorSystem1 manages confidential databases and ActorSystem2 provides offloading for applications.

their requirements and expectations. In order to show flexibility of the framework in defining required policy for different users, we consider three types of users: i) a user without specific privacy concerns ii) a privacy-concerned user iii) an extremely cautious user with utmost privacy concerns.

Figure 7.5 showed different modules involved in *Image* application. The right column of the figure are services provided by the cloud spaces. These services are governed by the organization-wide hard policy that we previously defined. Based on this figure, *Image* application consists of components defining left column. These components include: image loader, face detector, feature extractor, and face recognizer. Thus, application soft-policy should govern these components for different types of users.

For type i users without additional privacy concerns, *Image* application components can be arbitrary placed at different locations ranging from mobile device to public or private cloud spaces. Figure 7.10 shows the overall structure of *Image* application for these users. The only restriction for these types of users is to keep the image loader component on the mobile device, as photos are locally stored/taken on the mobile device.

Table 7.7 shows the required soft policy file for type i end-users, users without specific

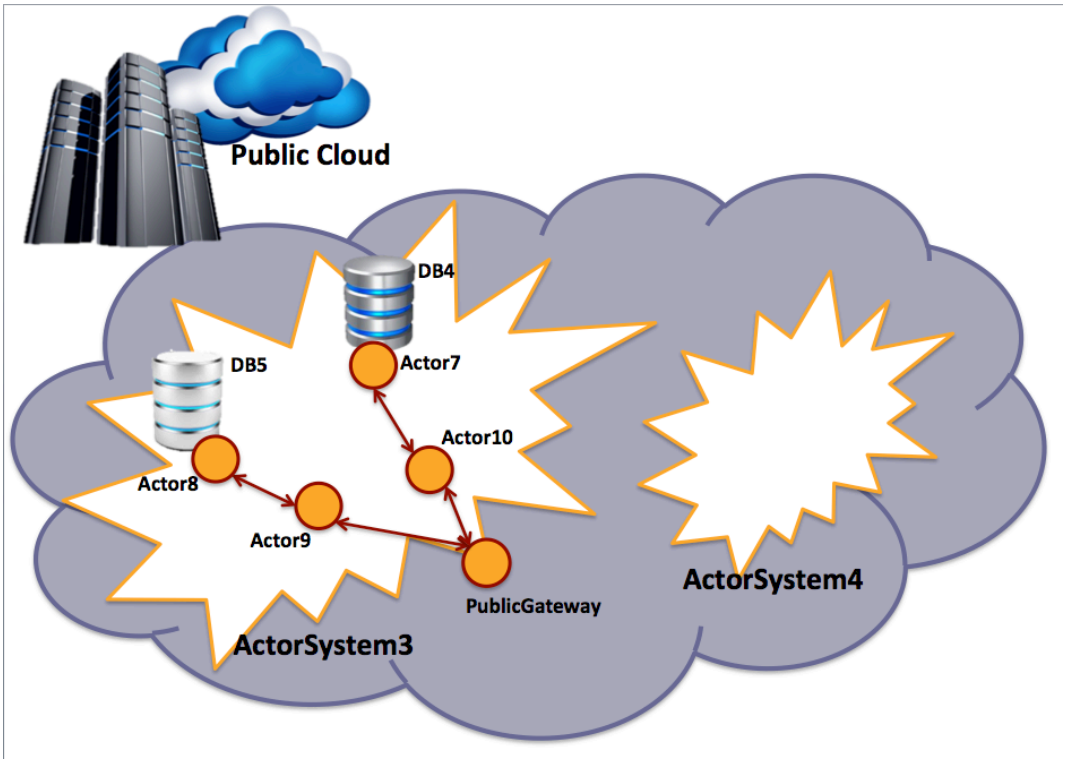


Figure 7.9: Structure of the public cloud space. The structure of the public cloud space consisting of 2 ActorSystems. ActorSystem3 manages 2 public databases and ActorSystem4 provides offloading for applications.

privacy concerns. It first defines user actor-system, line 1, and then defines the required entities, lines 2, 3, 4, and 5. Optionally, developers decided to restrict creation of any other actor or actor-systems on user device, lines 6 and 7. Note that these two lines are optional and can be changed for different applications based on developer preferences. The remaining part is to define rules restricting actions allowed in the application and its interactions with the private and public cloud spaces. Entire application components are first isolated from interacting with any other component, line 8, but then this restriction is relaxed by allowing all actors in user actor-system to interact with each other, line 9, in addition to allowing face recognizer actor to communicate with private and public gateway actors, line 10. Although access to private databases is restricted for different types of users, we do not need to worry about having such restrictions added in soft policy as well. Since such restrictions are already defined as part of the hard policy, only users with proper access rights can access private databases. For all other users, requests to access private databases will be rejected by the gateway actor according to the hard policy defined for that application instance. In order to allow face recognizer, feature extractor, and face detector components to be offloaded to public and private cloud spaces, additional rule is defined, line 11. Finally, required rule

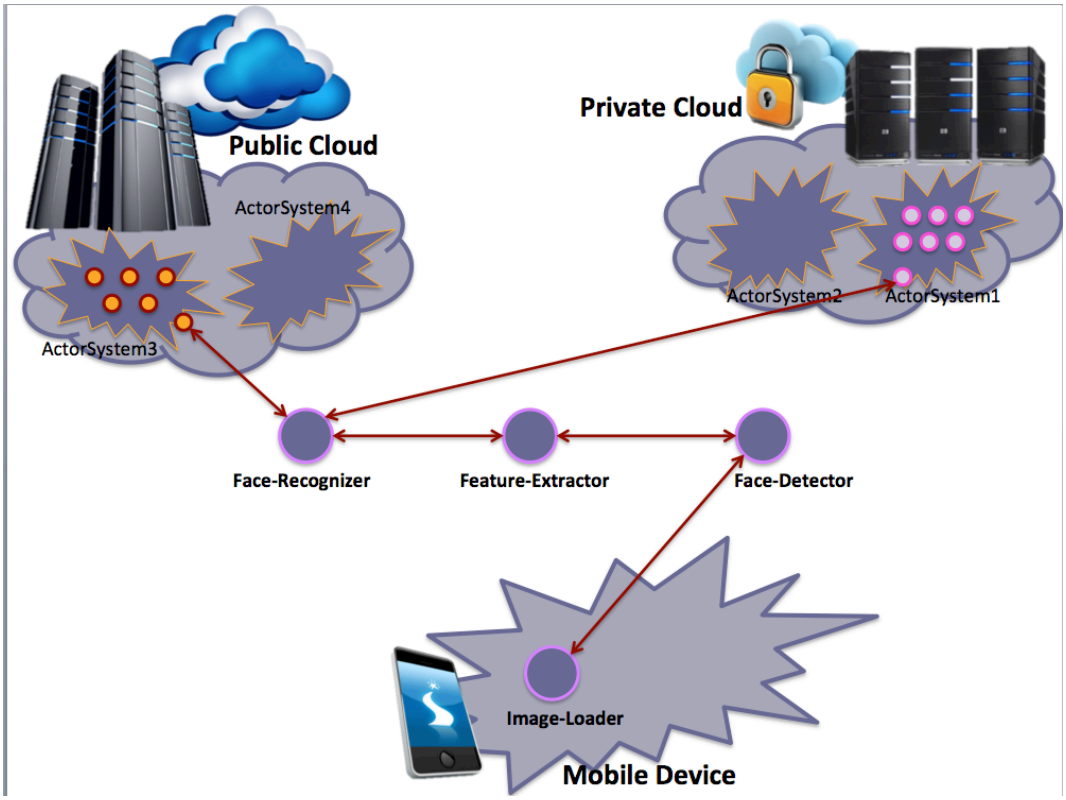


Figure 7.10: Application for users without specific privacy concerns. The structure of *Image* application for users without specific privacy concerns. Application components, face-recognizer, feature-extractor, and face-detector, can be placed on any of mobile device or different cloud offloading spaces.

orderings are defined to prioritize defined rules, lines 12, 13, and 14.

For type ii users with privacy concerns, *Image* application components containing privacy-sensitive information must remain on the phone. These components include image loader, face detector, and feature extractor actors that deal with original user-submitted picture containing real faces. Figure 7.11 shows the overall structure of *Image* application for these users. The only remaining component that can be offloaded for these users is face recognizer. Face recognizer works with extracted features from the original picture and does not contain any real face. Thus, it is safe to assume that it can be offloaded.

Table 7.8 shows the required soft policy file for type ii end-users, users with privacy concerns. It can be seen that it is exactly the same policy as Table 7.7 with the only modification in the rule defining off-loadable components, line 11. Instead of allowing all components to use private and public cloud spaces, we limit the use of those spaces to face recognizer component.

For type iii users with utmost privacy concerns, *Image* application components containing

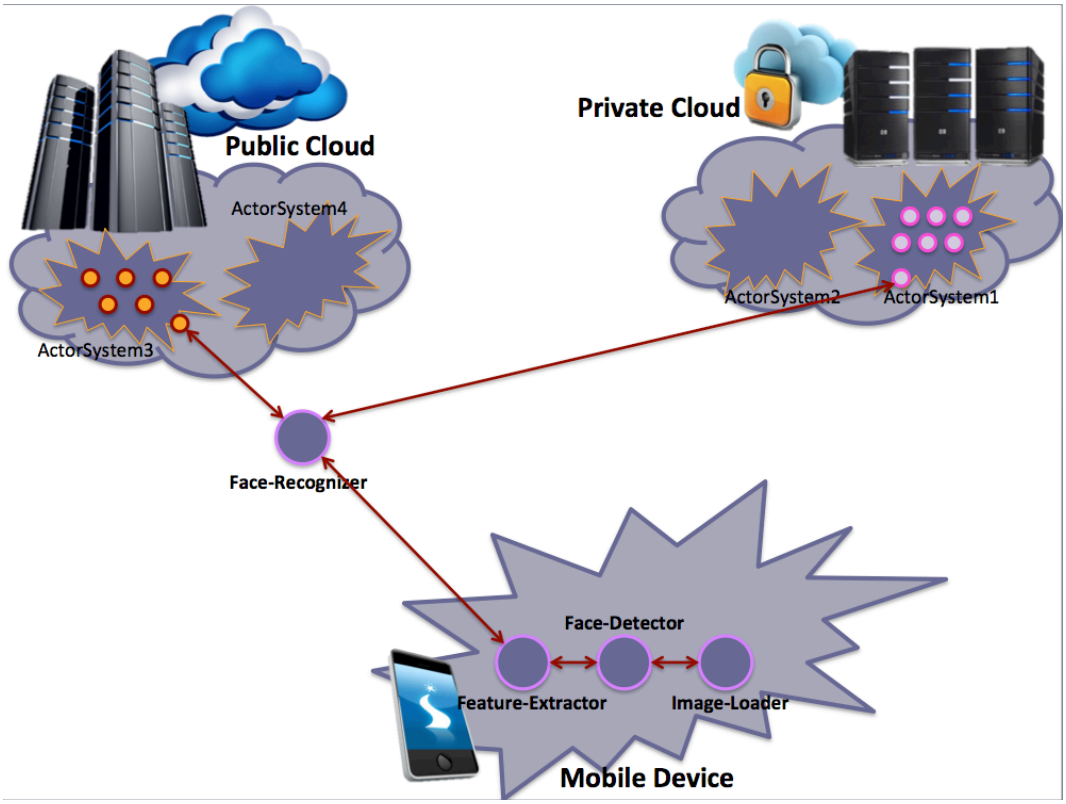


Figure 7.11: Application for privacy-concerned users. The structure of *Image* application for privacy-concerned users. Application components, image loader, face-detector, and feature-extractor, must remain on the phone, as they process original picture containing real faces. Face recognizer component uses extracted features, does not need any real face picture, and can be offloaded.

privacy-sensitive information must remain on the phone. In addition, face recognizer component must also remain on the mobile device and not offloaded. Although extracted features can not be used to reconstruct the original image, they still reveal partial information about the content of the original image. As a result, extremely cautious users might want to prevent offloading of these components as well. Figure 7.12 shows the overall structure of *Image* application for these users. It can be seen that all application components must remain on the phone and only reference images are received from remote cloud spaces.

Table 7.9 shows the required soft policy file for type iii end-users, extremely cautious users with utmost privacy concerns. The policy is the same as policies defined in Table 7.7 and Table 7.8 with the only modification that the rule allowing offloading of some of the components is removed. Instead of allowing all components to use private and public cloud spaces, we limit their execution to local mobile device.

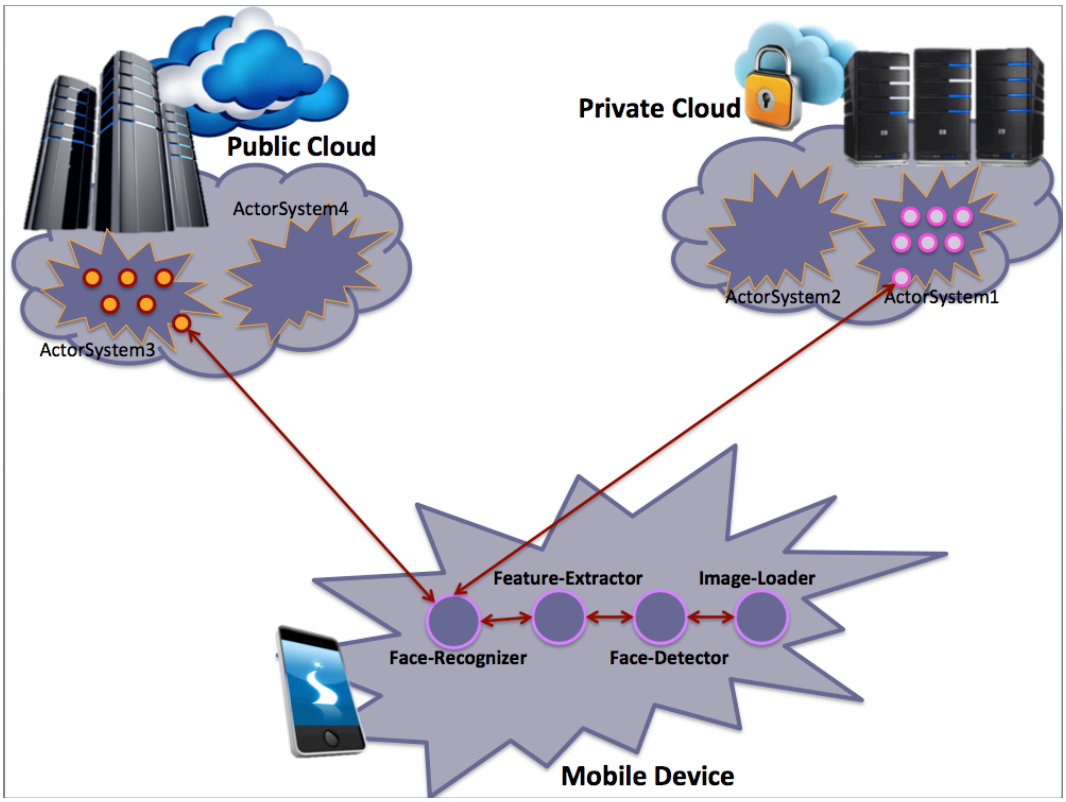


Figure 7.12: Application for extremely cautious users. The structure of *Image* application for extremely cautious users with utmost privacy concerns. All application components must remain on the mobile device.

7.2 Defining Execution Quality for Different Components at Various Locations

The grammar defined for authorization policy definition allows defining additional restrictions in terms of entity-based restrictions. One application of that can be to define different quality of service for different resources. As a result, application components can be executed at different locations depending on the requested quality of service for that location. Let's consider the *Image* application. for this application, quality of the service can be defined in two different ways: i) use different number of known faces for recognizing step ii) use different picture resolution and size for face detection and feature extraction. Using the first approach results in less comparison time to recognize the faces in input picture. In order to detect faces or extract features, initial picture is considered at different sizes usually differing by a factor of 2. Usually the maximum size is the original picture size. However, having quality of service defined based on picture size, the maximum size can be limited to lower sizes. Lower picture size results in faster processing time for both face detection and feature

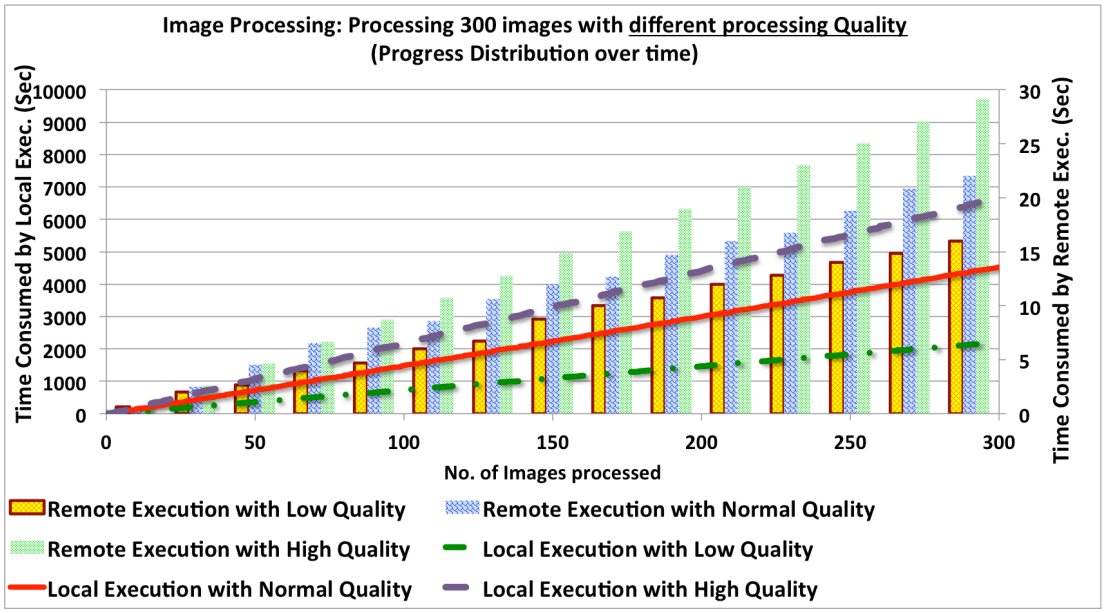


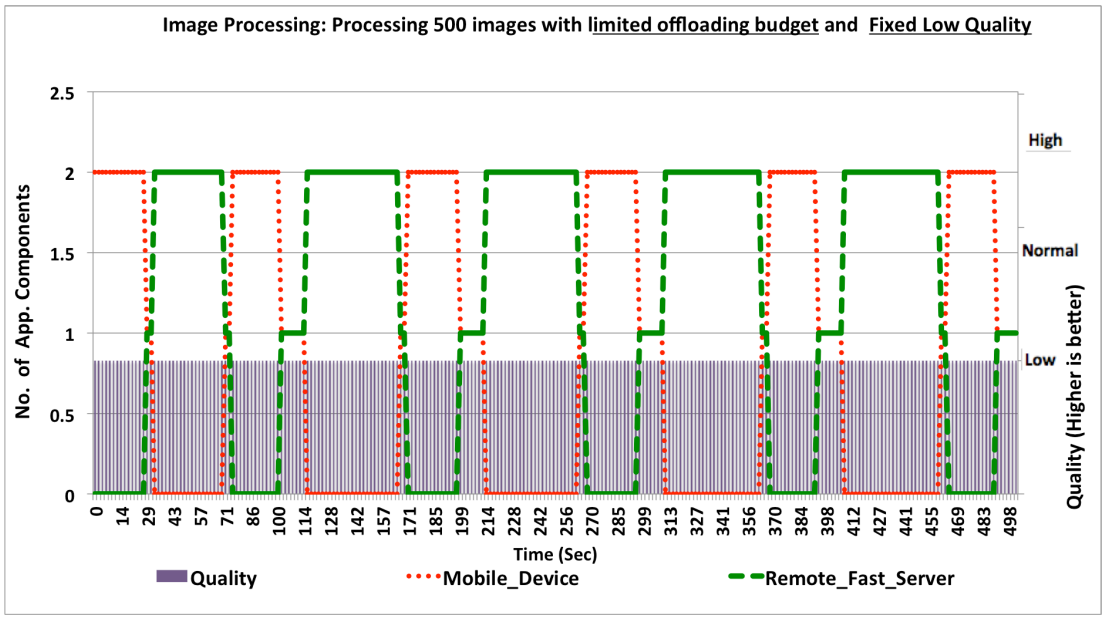
Figure 7.13: Local and remote execution of Image processing with different fixed qualities.

extraction. However, it also lowers the accuracy and quality of the result.

In order to show the effect of quality of processing service on the performance, we conducted an experiment running image processing application for processing 300 images both locally and remotely with different selected qualities. Figure 7.13 shows the result. It can be seen that the required time to process images at different qualities is proportional to the selected quality of service. The difference is much more severe when processing images locally on the phone, as phone processor is much slower than the remote laptop and the effect of any change to the amount of work becomes much more visible.

When different quality of services are available for component execution, the framework is allowed to pick any quality of service for component execution at any location. Lower quality of service will lower required processing. As a result, when offloading goal is defined based on maximizing application performance or minimizing mobile energy consumption, the system always picks the lowest quality of service for components regardless of the location. This is because of the fact that no additional value is defined for higher quality of service. One solution is to define an additional value for higher quality of service and allow the system to pick the quality of service to reach a certain added value. Another solution is to relate quality of service to other parameters. A good example is relating quality of service to different locations. In this scenario, we specify a certain quality of service for different components at different locations. Offloading decision making results in the optimal component distribution plan to meet a certain target goal. Based on this offloading

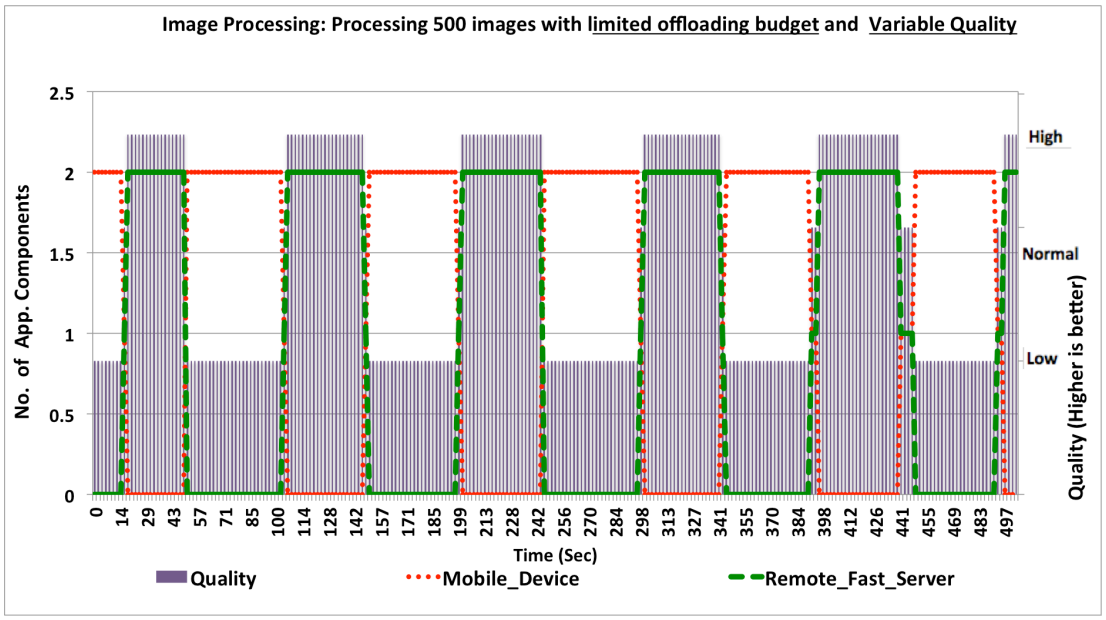
Figure 7.14: Code offloading with fixed quality of service for all locations and limited offloading budget rate. Mobile code offloading to a single remote server with fixed quality of service and limited offloading budget rate. Quality is set to Low for all locations



plan, different quality of service can be selected for different components. Figure 7.14 shows offloading result for the *Image* application processing 300 images using two worker actors. The problem faces limited offloading budget rate while the quality of service is fixed at *low* for all components and at all locations. It can be seen that the components are started on the phone and then migrated to the remote server. When offloading budget for current time interval is reached, components are brought back to the mobile phone. With current time interval passing, offloading budget will be retested. As a result, components are offloaded to the remove server and this cycle continues. However, since we have fixed the quality of service for all components at all locations to *low*, all processing is performed at low resolution regardless of the location.

We can change the above scenario and define different quality of service for different locations. In this example, we specified *high* processing resolution for worker actors when processing images at the remote server and *low* processing resolution when processing images at local mobile device. This is consistent with the fact that mobile device is much slower than the remote server and processing images at a high quality on mobile device takes a lot longer. Figure 7.15 shows the result for this case. It can be seen that quality of execution is set at low when components are executed locally and at high when components are executed at the remote server. Since remote server is more-resourceful than local mobile device, increasing the quality of processing images on it has negligible effect on execution time. As

Figure 7.15: Code offloading with variable quality of service for different locations and limited offloading budget rate. Mobile code offloading to a single remote server with variable quality of service and limited offloading budget rate. Quality of the service is set to Low for local mobile device and high for fast remote server.



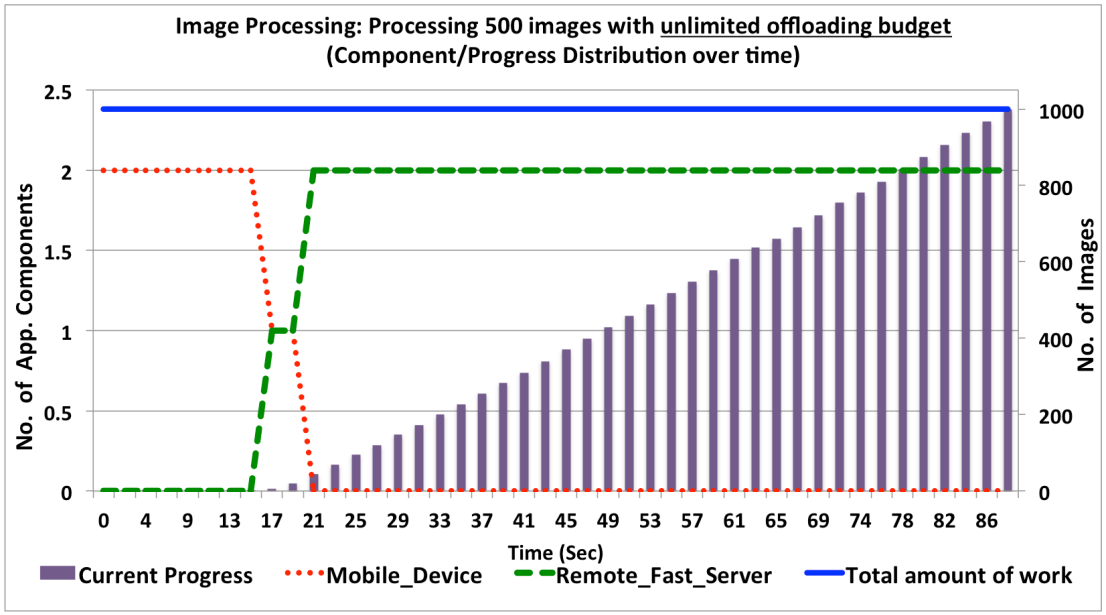
a result, it can be seen that both Figure 7.14 and Figure 7.15 takes same amount of execution time. However, the outcome from Figure 7.14 is at a higher quality and better accuracy as offloaded components are executed at a higher resolution.

7.3 Defining Offloading Budget Limitations for Different Cloud Resources

If third party cloud resources are used for offloading of an application, there will be an additional cost for using these resources. This cost is usually defined based on resource usage. As a result, many organizations, developers, or users might want to limit additional costs related to using third party cloud resources, public cloud, for offloading. Our framework allows defining an offloading budget for different resources, which will then be enforced as an additional constraints while solving the offloading optimization problem.

With our framework in use and without any budget limitation on using cloud resources, application components will be moved around and distributed between the mobile device and the remote cloud server in a way that optimized target offloading goal. Figure 7.16 shows a sample output for the *Image* application processing 500 images using a single remote cloud

Figure 7.16: Code offloading with unlimited offloading budget. Mobile code offloading to a single remote server without any offloading budget restrictions



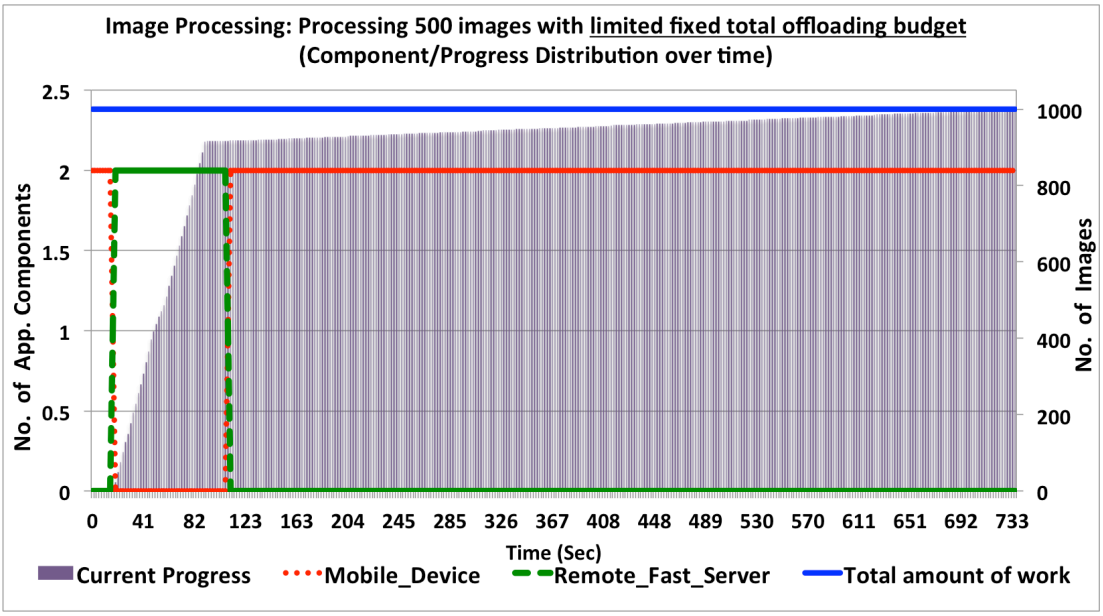
server without any offloading budget.

As can be seen in Figure 7.16, application consists of two worker components that are both started on the mobile device. When the application starts, the profiler begins to collect application and resource usage for different components. A few seconds after the start of the application, elasticity manager asks local worker components to migrate to the remote cloud server, because the cloud server has the faster resources. It takes local workers a few seconds to migrate to the new location. When this happens, the progress significantly improves, as can be seen by background vertical bars. Since there is no budget limitation for using remote server, components remain on the remote server and process all the remaining pictures at a faster speed on the remote location.

Now, let's consider the case that we define a fixed total offloading budget for the use of remote server. This fixed budget is defined as a lump sum in terms of the time entire remote server processor is being used. It is relatively straight forward to define this cost in any other desired form, such as used time per core, used time per virtual machine, etc. Elasticity manager is free to use the remote server for code offloading up to the specified budget. Figure 7.17 shows the output for the same previous *Image* application example subject to a fixed total offloading budget.

Figure 7.17 shows that a few seconds after the start of the application, the two local workers are migrated to the remote server in order to improve the overall performance. Workers will remain on the remote server for several seconds until the fixed total offloading

Figure 7.17: Code offloading with a fixed total offloading budget. Mobile code offloading to a single remote server with a fixed total offloading budget

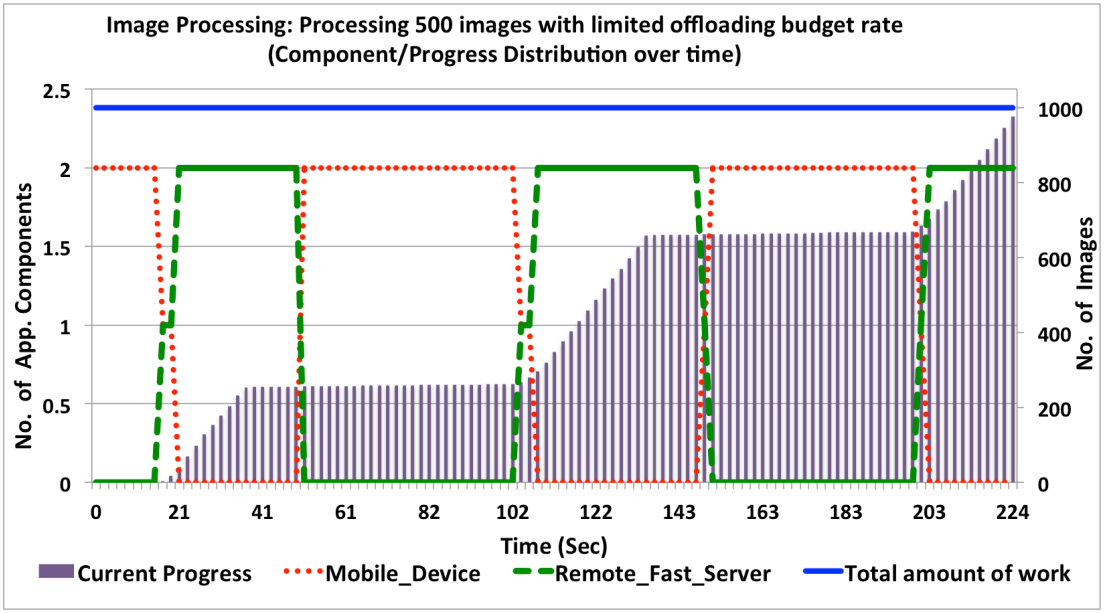


budget for using the remote server is reached. At this time, workers are brought back to the mobile device and continue with processing the remaining pictures on the local mobile device. This is also reflected in the vertical bars representing progress of processing pictures. When workers are placed on the fast remote server, the slope of the vertical bar increases become very steep. However, when workers are brought back to mobile device, this steep slop will become much flatter due to lower availability of resources on the mobile device. Comparing the total execution time of Figure 7.16 and Figure 7.17 reveals that having restriction on offloading budget can significantly affect the performance and increase the total execution time.

While defining a total fixed offloading budget can be useful in many applications, limiting offloading budget rate can also be beneficial. Offloading budget rate is defined as a fixed budget per time interval and repeated when that interval is passed. A good example is defining a fixed budget for offloading per hour. This ensures that offloading budget will not exceed a certain fixed amount per hour but will be restarted when the first hour is passed, if the budget is consumed. Figure 7.18 shows the output for the previous *Image* application example subject to a limited offloading budget rate, which in this case is defined per minute.

As can be seen in Figure 7.18, local workers are started on the mobile device but then offloaded to the remote server after a few seconds. They remain on the remote server till the offloading budget for current time interval is consumed. As a result, the elasticity manager brings back the offloaded components to the mobile device and they are executed locally till

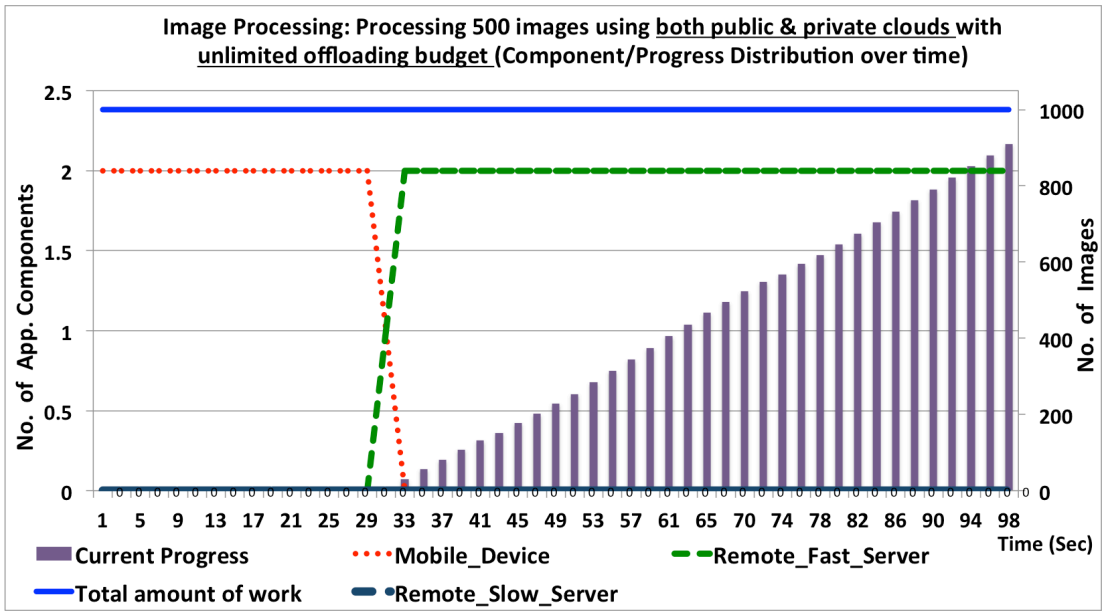
Figure 7.18: Code offloading with a limited offloading budget rate. Mobile code offloading to a single remote server with a fixed offloading budget per time interval (in this case every 1 minutes)



the current time interval is finished. When current time interval is passed, the offloading budget is refreshed. Thus, worker components are offloaded again to the remote server and this cycle continues till all images are processed. Comparing the total execution time of Figure 7.18 and Figure 7.17 reveals that having a renewable offloading budget significantly improves the performance.

Due to additional costs resulting from using third party cloud resources, many organizations or developers might want to use a hybrid cloud consisting of both public and private cloud spaces. IMCM is flexible enough to allow definition of different offloading budget restrictions for various cloud resources. Considering a hybrid cloud consisting of a public cloud space, with more resources to use due to public cloud elasticity, and a private cloud space, with less resources to use due to limitations with availability of in-house resources, and the previous *Image* application, we can have the following situations. The basic case happens when we have no limitation on using public cloud. Note that we are assuming the public cloud is able to provide a faster processing time compared to private cloud space, as a public cloud may have better hardware or more elasticity in terms of adding additional resources whenever needed. In contrast, a private cloud space has a fixed number of in-house servers owned by the organization but shared between different users and applications. Due to a fixed number of servers in a private cloud, there is limitation in terms of elasticity of the cloud and the performance degrades when the number of users or applications increases.

Figure 7.19: Code offloading with unlimited offloading budget in a Hybrid Cloud. Mobile code offloading to a hybrid cloud without any offloading budget restrictions. Hybrid cloud consists of a more-resourceful public cloud space and a more-restricted private cloud.



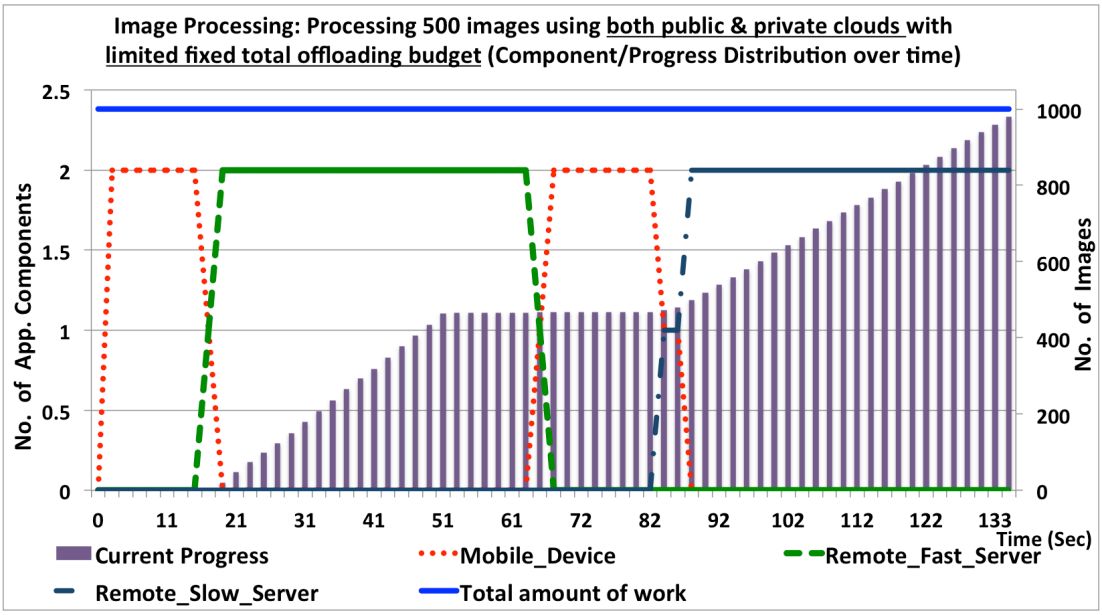
Thus, it is reasonable to assume that a public cloud provides better performance when compared to a private cloud as we do for this example. Without any budget limitation on using public and private clouds, results from running the *Image* application can be summarized as Figure 7.19.

The results from Figure 7.19 shows that after application is started locally on the mobile device, worker components are migrated to the faster public cloud and the remaining of the computation is performed on the public cloud. Since we have no budget limitation in terms of using cloud resources and public cloud has more resources with improved performance, no component is sent to the private cloud and all components are placed at public cloud space.

In order to control offloading cost, we can define a fixed total offloading budget for the use of more-resourceful public cloud. Note that we are assuming unlimited offloading budget for the private cloud, as it is owned by the organization and not subject to additional usage costs. Figure 7.20 shows the result for this case:

The result from Figure 7.20 shows that after starting the application on the mobile device, both worker components are migrated to the faster more-resourceful public cloud. After a while, the fixed budget for using public cloud space is consumed and the components are brought back to the mobile device. Since there is no more budget left for using the public cloud, worker components are offloaded to the slower but free private cloud server. Worker components remains on the private cloud server and process the remaining images at a faster

Figure 7.20: Code offloading with fixed total offloading budget in a Hybrid Cloud. Mobile code offloading to a hybrid cloud with a fixed total offloading budget restriction for use of public cloud space. Hybrid cloud consists of a budgeted more-resourceful public cloud space and an unlimited more-restricted private cloud.



speed than mobile device, but slower than the more-resourceful public cloud server.

It is also possible to define a fixed budget rate for the use of public cloud space. Figure 7.21 shows the output result for *Image* application subjected to a fixed offloading budget rate for use of more-resourceful public cloud space and unlimited offloading budget for the use of more-restricted private cloud space.

Application components are started initially on the mobile device and offloaded to the fastest remote space, in this case public cloud, within a few seconds. Worker components remains on the public cloud server til the offloading budget rate is consumed for that time interval. They are, then, brought back to the phone and then offloaded to the more-restricted private cloud. With the current time interval passing, public cloud usage budget is restarted. As a result, components are migrated from the private cloud server to the faster more-resourceful public cloud and this cycle continues till all images are processed. These examples show the flexibility of the proposed framework in defining additional constraints in terms of offloading budget for different cloud resources.

Figure 7.21: Code offloading with fixed offloading budget rate in a Hybrid Cloud. Mobile code offloading to a hybrid cloud with a fixed offloading budget rate restriction for use of public cloud space. Hybrid cloud consists of a budget-rate-limited more-resourceful public cloud space and an unlimited more-restricted private cloud.

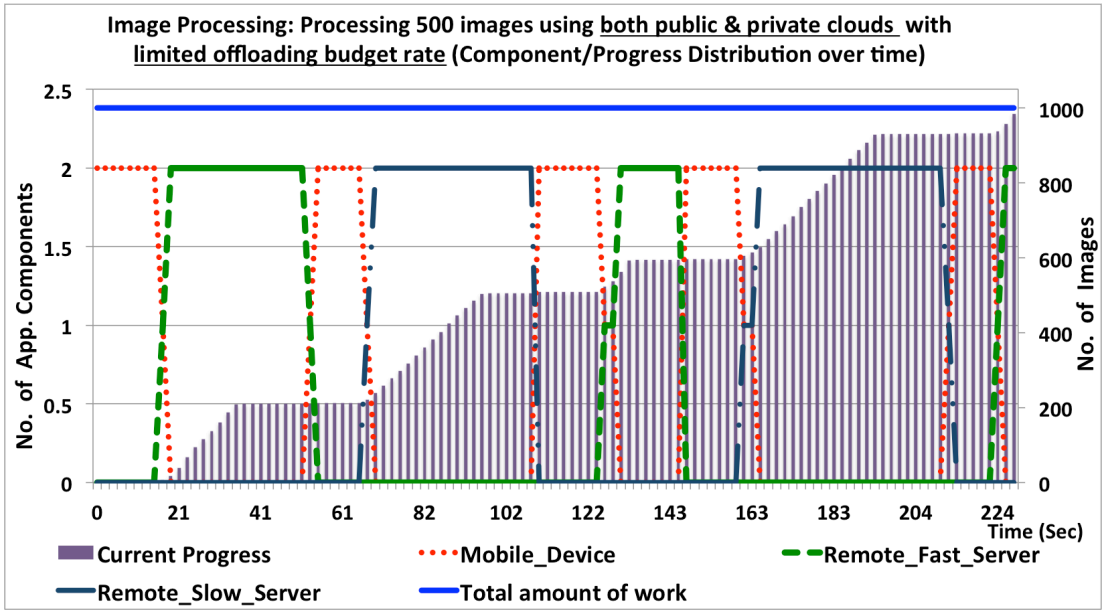


Table 7.4: Private cloud policy defined as part of the hard policy file

1. ActorSystem: {Name:ActorSysPrivate1, Static (URL:174.123.78.456, Port:1362))}
2. Actor: {Name:ActorPrivateGate, Static (
   Reference: akka.tcp://app@174.123.78.456/privateGateway,
   ActorSystem:"ActorSysPrivate1)}
3. Actor: {Name:ActorPrivateVisasDB, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor1,
   ActorSystem:ActorSysPrivate1)}
4. Actor: {Name:ActorPrivateResidentDB, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor2,
   ActorSystem:ActorSysPrivate1)}
5. Actor: {Name:ActorPrivateCitizenDB, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor3,
   ActorSystem:ActorSysPrivate1)}
6. Actor: {Name:ActorPrivateVisaProcessor, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor4,
   ActorSystem:ActorSysPrivate1)}
7. Actor: {Name:ActorPrivateResidentProcessor, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor5,
   ActorSystem:ActorSysPrivate1)}
8. Actor: {Name:ActorPrivateCitizenProcessor, Static (
   Reference: akka.tcp://app@174.123.78.456/Actor6,
   ActorSystem:ActorSysPrivate1)}
9. AnonymousActors: {Name:AnonymousPrivate1,
   Ref-ActorSystem:ActorSysPrivate1, Existence:FORBIDDEN}
10. AnonymousActorSystems: {Name: Other-ActorSys-Private,
   URL:174.123.78.456, Creation: FORBIDDEN}
11. Rule: {Name:Private-Rule-1, Subject (ActorSystems:ActorSysPrivate1),
   Object (ALL), Actions: ALL, Permission: DISALLOWED}
12. Rule: {Name:Private-Rule-2, Subject (ActorSystems:ActorSysPrivate1),
   Object (ActorSystems:ActorSysPrivate1),
   Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
13. Rule: {Name:Private-Rule-3, Subject (Actor:ActorPrivateGate),
   Object (ALL), Actions: SEND-TO, RECEIVE-FROM,
   Permission: ALLOWED}
14. Rule-Order: {Name: Private-Rule-Order-1, Subject (Rules: Private-Rule-2),
   Object(Rules: Private-Rule-1), Order: PRECEDENCE}
15. Rule-Order: {Name: Private-Rule-Order-2, Subject (Rules: Private-Rule-3),
   Object(Rules: Private-Rule-1), Order: PRECEDENCE}
16. ActorSystem: {Name:ActorSysPrivate2, Static (URL:174.123.78.456, Port:1369))}
17. AnonymousActors: {Name:AnonymousPrivate2,
   Ref-ActorSystem:ActorSysPrivate2, Existence:ALLOWED}
18. Rule: {Name:Private-Rule-4, Subject (ActorSystems:ActorSysPrivate2),
   Object (ALL), Actions: ALL, Permission: ALLOWED}

Table 7.5: Public cloud policy defined as part of the hard policy file

19. ActorSystem: {Name:ActorSysPublic1, Static (URL:179.987.654.321, Port:1352))}
20. Actor: {Name:ActorPublicGate, Static (
    Reference: akka.tcp://app@179.987.654.321/publicGateway,
    ActorSystem:"ActorSysPublic1)}
21. Actor: {Name:ActorPublicInternationalDB, Static (
    Reference: akka.tcp://app@179.987.654.321/Actor7,
    ActorSystem:ActorSysPublic1)}
22. Actor: {Name:ActorPublicNationalDB, Static (
    Reference: akka.tcp://app@179.987.654.321/Actor8,
    ActorSystem:ActorSysPublic1)}
23. Actor: {Name:ActorPublicInternationalProcessor, Static (
    Reference: akka.tcp://app@179.987.654.321/Actor10,
    ActorSystem:ActorSyspublic1)}
24. Actor: {Name:ActorPublicNationalProcessor, Static (
    Reference: akka.tcp://app@179.987.654.321/Actor9,
    ActorSystem:ActorSysPrivate1)}
25. AnonymousActors: {Name:AnonymousPublic1,
    Ref-ActorSystem:ActorSysPublic1, Existence:FORBIDDEN}
26. AnonymousActorSystems: {Name: Other-ActorSys-Public,
    URL:179.987.654.321, Creation: FORBIDDEN}
27. Rule: {Name:Public-Rule-1, Subject (ActorSystems:ActorSysPublic1),
    Object (ALL), Actions: ALL, Permission: DISALLOWED}
28. Rule: {Name:Public-Rule-2, Subject (ActorSystems:ActorSysPublic1),
    Object (ActorSystems:ActorSysPublic1),
    Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
29. Rule: {Name:Public-Rule-3, Subject (Actor:ActorPublicGate),
    Object (ALL), Actions: SEND-TO, RECEIVE-FROM,
    Permission: ALLOWED}
30. Rule-Order: {Name: Public-Rule-Order-1, Subject (Rules: Public-Rule-2),
    Object(Rules: Public-Rule-1), Order: PRECEDENCE}
31. Rule-Order: {Name: Public-Rule-Order-2, Subject (Rules: Public-Rule-4),
    Object(Rules: Public-Rule-1), Order: PRECEDENCE}
32. ActorSystem: {Name:ActorSysPublic2, Static (URL:179.987.654.321, Port:1359))}
33. AnonymousActors: {Name:AnonymousPublic2,
    Ref-ActorSystem:ActorSysPublic2, Existence:ALLOWED}
34. Rule: {Name:Public-Rule-4, Subject (ActorSystems:ActorSysPublic2),
    Object (ALL), Actions: ALL, Permission: ALLOWED}

Table 7.6: End-user application policy defined as part of the hard policy file. It uses dynamic binding property of the framework to support any arbitrary architecture of to-be-developed applications.

35. AnonymousActorSystems: {Name: Arbitrary-ActorSys-User, ANONYOUS-URLs, Creation: ALLOWED, ActorSystem-Limitation: "UNLIMITED", Actors-per-ActorSystem-Limitation: "UNLIMITED" }

Table 7.7: End-user soft policy for users without specific privacy concerns

1. ActorSystem: {Name:ActorSysUser, Static (URL:98.123.123.456, Port:1979))}
2. Actor: {Name:UserImageLoader, Static (
   Reference: akka.tcp://app@98.123.123.456/Image-Loader,
   ActorSystem:"ActorSysUser)}
3. Actor: {Name:UserFaceDetector, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Detector,
   ActorSystem:"ActorSysUser)}
4. Actor: {Name:UserFeatureExtractor, Static (
   Reference: akka.tcp://app@98.123.123.456/Feature-Extractor,
   ActorSystem:"ActorSysUser)}
5. Actor: {Name:UserFaceRecognizer, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Recognizer,
   ActorSystem:"ActorSysUser)}
6. AnonymousActors: {Name:AnonymousUser,
   Ref-ActorSystem:ActorSysUser, Existence:FORBIDDEN}
7. AnonymousActorSystems: {Name: Other-ActorSys-User,
   URL:98.123.123.456, Creation: FORBIDDEN}
8. Rule: {Name:User-Soft-Rule-1, Subject (ActorSystems:ActorSysUser),
   Object (ALL), Actions: ALL, Permission: DISALLOWED}
9. Rule: {Name:User-Soft-Rule-2, Subject (ActorSystems:ActorSysUser"),
   Object (ActorSystems: "ActorSysUser"),
   Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
10. Rule: {Name:User-Soft-Rule-3, Subject (Actors:UserFaceRecognizer"),
   Object (Actors:ActorPublicGate, ActorPrivateGate),
   Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
11. Rule: {Name:User-Soft-Rule-4,
   Subject (Actors:UserFaceDetector, UserFeatureExtractor,
      UserFaceRecognizer),
   Object (ActorSystems: "ActorSysUser, ActorSysPrivate2,
      ActorSysPublic2"),
   Actions: ALL, Permission: ALLOWED}
12. Rule-Order: {Name: User-Soft-Rule-Order-1, Subject (Rules: User-Soft-Rule-2),
   Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}
13. Rule-Order: {Name: User-Soft-Rule-Order-2, Subject (Rules: User-Soft-Rule-3),
   Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}
14. Rule-Order: {Name: User-Soft-Rule-Order-3, Subject (Rules: User-Soft-Rule-4),
   Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}

Table 7.8: End-user soft policy for privacy-concerned users

1. ActorSystem: {Name:ActorSysUser, Static (URL:98.123.123.456, Port:1979))}
2. Actor: {Name:UserImageLoader, Static (
   Reference: akka.tcp://app@98.123.123.456/Image-Loader,
   ActorSystem:"ActorSysUser)}
3. Actor: {Name:UserFaceDetector, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Detector,
   ActorSystem:"ActorSysUser)}
4. Actor: {Name:UserFeatureExtractor, Static (
   Reference: akka.tcp://app@98.123.123.456/Feature-Extractor,
   ActorSystem:"ActorSysUser)}
5. Actor: {Name:UserFaceRecognizer, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Recognizer,
   ActorSystem:"ActorSysUser)}
6. AnonymousActors: {Name:AnonymousUser,
   Ref-ActorSystem:ActorSysUser, Existence:FORBIDDEN}
7. AnonymousActorSystems: {Name: Other-ActorSys-User,
   URL:98.123.123.456, Creation: FORBIDDEN}
8. Rule: {Name:User-Soft-Rule-1, Subject (ActorSystems:ActorSysUser),
   Object (ALL), Actions: ALL, Permission: DISALLOWED}
9. Rule: {Name:User-Soft-Rule-2, Subject (ActorSystems:ActorSysUser"),
   Object (ActorSystems: "ActorSysUser"),
   Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
10. Rule: {Name:User-Soft-Rule-3, Subject (Actors:UserFaceRecognizer"),
    Object (Actors:ActorPublicGate, ActorPrivateGate),
    Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
11. Rule: {Name:User-Soft-Rule-4,
    Subject (Actors:UserFaceRecognizer),
    Object (ActorSystems: "ActorSysUser, ActorSysPrivate2,
    ActorSysPublic2"),
    Actions: ALL, Permission: ALLOWED}
12. Rule-Order: {Name: User-Soft-Rule-Order-1, Subject (Rules: User-Soft-Rule-2),
    Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}
13. Rule-Order: {Name: User-Soft-Rule-Order-2, Subject (Rules: User-Soft-Rule-3),
    Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}
14. Rule-Order: {Name: User-Soft-Rule-Order-3, Subject (Rules: User-Soft-Rule-4),
    Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}

Table 7.9: End-user soft policy for extremely cautious users with utmost privacy concerns

1. ActorSystem: {Name:ActorSysUser, Static (URL:98.123.123.456, Port:1979))}
2. Actor: {Name:UserImageLoader, Static (
   Reference: akka.tcp://app@98.123.123.456/Image-Loader,
   ActorSystem:"ActorSysUser)}
3. Actor: {Name:UserFaceDetector, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Detector,
   ActorSystem:"ActorSysUser)}
4. Actor: {Name:UserFeatureExtractor, Static (
   Reference: akka.tcp://app@98.123.123.456/Feature-Extractor,
   ActorSystem:"ActorSysUser)}
5. Actor: {Name:UserFaceRecognizer, Static (
   Reference: akka.tcp://app@98.123.123.456/Face-Recognizer,
   ActorSystem:"ActorSysUser)}
6. AnonymousActors: {Name:AnonymousUser,
   Ref-ActorSystem:ActorSysUser, Existence:FORBIDDEN}
7. AnonymousActorSystems: {Name: Other-ActorSys-User,
   URL:98.123.123.456, Creation: FORBIDDEN}
8. Rule: {Name:User-Soft-Rule-1, Subject (ActorSystems:ActorSysUser),
   Object (ALL), Actions: ALL, Permission: DISALLOWED}
9. Rule: {Name:User-Soft-Rule-2, Subject (ActorSystems:ActorSysUser"),
   Object (ActorSystems: "ActorSysUser"),
   Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
10. Rule: {Name:User-Soft-Rule-3, Subject (Actors:UserFaceRecognizer"),
    Object (Actors:ActorPublicGate, ActorPrivateGate),
    Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}
11. Rule-Order: {Name: User-Soft-Rule-Order-1, Subject (Rules: User-Soft-Rule-2),
    Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}
12. Rule-Order: {Name: User-Soft-Rule-Order-2, Subject (Rules: User-Soft-Rule-3),
    Object(Rules: User-Soft-Rule-1), Order: PRECEDENCE}

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this chapter we provide a summary of the dissertation. We will also discuss its limitations and future directions.

8.1 Summary

In this dissertation we studied the problem of code offloading for mobile hybrid cloud applications. Optimizing application performance or mobile energy consumption are two important target offloading goals for many mobile applications. We therefore focus on developing offloading decision-making models for these two target offloading goals in the face of varying application requirements, environmental conditions, and user expectations. We follow four main principles in our proposed solution: *separation of concerns*, *flexibility*, *ease of use*, and *adaptivity*. The result is *IMCM*: Illinois Mobile Cloud Management middleware framework for mobile hybrid cloud application development.

In the design of *IMCM*, we consider applications as a combination of independent components storing data or processing code. We use the Actor model of computation to present code and data components and their interactions. Actors are concurrent objects interacting via asynchronous message passing. Our proposed framework provides a systematic method to support dynamic application component configuration and distribution for mobile cloud applications based on run-time parameters and individual application requirements. Specifically, we design an approach using specification of application-defined requirements and user-defined expectations. Application target goals are significantly different and range from maximizing application performance to minimizing mobile energy consumption. We model each mobile application according to its target goal and formulate the application into an optimization problem. Every possible configuration of components that satisfy the requirements will be evaluated and the best distribution plan that optimizes the dynamic application target goal is selected and enforced.

The architecture of *IMCM* has three main parts: first, a light-weight monitoring sys-

tem, called *Monitor*, to capture dynamic environmental parameters and end-user context, profile application resource usage and communications, as well as monitoring availability and performance of cloud resources. Profiling energy consumption per specific application components is primary of importance and requires design and development of a fine-grained automatic energy consumption model, as most mobile devices do not provide any tool for direct measurement of consumed energy and different applications with arbitrary number of components might be running at any time.

Second, we design and implement two independent performance-based and energy-based models to enable transparent automatic configuration and distribution of application code and data components that address specific organization, application, and end-user requirements. These models leverage dynamic information from the *Monitor* on run-time parameters, energy and resource usage of different components, and application characteristics to optimize application performance or mobile energy consumption with respect to a predefined policy. While offloading to a single remote server and serial monotonic application execution results in similar offloading plan for both target goals of optimizing for application performance and mobile energy consumption, supporting hybrid cloud environment with multiple public and private cloud spaces in addition to fully parallel application execution results in significantly different component distribution plan for different offloading goals. This requires design and development of a very different offloading model.

Finally, we design and develop the required grammar and enforcement mechanism to allow organizations, developers or end-users to easily adjust target offloading goal at run-time in addition to define required policy-based restrictions in terms of privacy, component move around, communications, resource accesses, offloading budget, and quality of service of resources. Privacy of data is a challenging issue especially when parts of the application are outsourced. Many companies are using a hybrid cloud model in order to keep the confidential or sensitive algorithms or user data within secure private in-house servers. The grammar of the policy specification language we develop allows organizations, application developers, and end users to define their required privacy authorization rules and adjust them during execution. A light-weight action control and policy management system is designed and implemented to interpret the defined policy rules and enforce them at run-time at different levels.

IMCM facilitates mobile hybrid cloud application development by allowing transparent and automatic distribution of application components between mobile device and multiple cloud spaces. It also detects run-time properties and adjust offloading plan according to dynamic environment. The actor model in turn allows dynamic reconfiguration and parametrization of the components based on application requirements, environmental con-

ditions, user expectations, and policy-based defined restrictions. The approach taken in *IMCM* is vastly applicable to other areas of distributed systems. For example, this framework can be used in Wireless Sensor Networks (WSNs) to allow dynamic configuration and distribution of services among different wireless nodes.

8.2 Limitations and Future Work

We try to ensure *conclusion validity* of our evaluation by checking the statistical significance of measured execution times with a robust non-parametric test at a relatively high level $\alpha = 0.01$. One threat to the *construct validity* of our experiment is the use of performance speedup and energy-saving factor as effectiveness metrics. With the amount of work increased, the gap between local mobile execution and other form of execution becomes larger. This reflects the improved performance and can be used to evaluate the performance of one application with different settings. However, the amount of work performed by different applications varies significantly. Moreover, different applications have different behavior, architecture and characteristics. Thus, comparison of speedup or energy-saving factor between different applications cannot be performed. The *external validity* of our evaluation is threatened by our focused corpus. Despite the fact that programs selected include benchmarks used in previous works, the corpus does not constitute a random sample of programs. Consequently, our results may generalize poorly. A larger study would mitigate this risk and is considered as future work.

Mobile hybrid cloud applications impose unique requirements that are crucial to their successful execution. We therefore focus on optimizing target offloading goal while satisfying application constraints in the design of *IMCM*. Our offloading decision-making models covers target offloading goals of maximizing application performance and minimizing mobile energy consumption. These decision-making models can be extended to cover other target goals, such as minimizing network data usage, minimizing interaction time, maximizing security, minimizing monetary cost of using cloud resources, etc. Moreover, multiple combination of these goals can be combined to generate new offloading goals for different applications.

IMCM assumes reliable network connection to remote cloud resources. Even with current reliable network connections, power outage or network failure still happens. As a result, reliability of the solution is a remaining challenge. Mobile devices rely on wireless or cellular network connections that are still limited in many locations, such as subway, tunnels, airplanes, undergrounds, and etc. The case of network connection disconnecting or cloud resources becoming unavailable during offloading still needs to be investigated. Caching a

copy of the offloaded components locally or on other cloud resources can solve this problem. However, the use of caching in a dynamic distributed system requires additional efforts, such as cache validation coherency, that needs to be further studied.

The adaptive middleware framework enforces a policy-based restriction on allowable actions and move-around of application components. Our work does not specify how these policies are defined and how effective they are in terms of providing privacy. With auditing solutions becoming popular, combing the framework with audition modules allow evaluation of enforced policy and potentially suggestions in terms of improving policy rules.

Another area of future work is optimizing when offloading decision-making should be executed and when the results should be enforced. Component distribution offloading updates can be distributed either periodically or in the event of specific occurrences. We followed a periodic approach where offloading decision-making and enforcement of the new plan are enforced at specific time intervals. We leave a through investigation of this approach for future work.

REFERENCES

[1] 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.

[2] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, *Agile application-aware adaptation for mobility*. ACM, 1997, vol. 31, no. 5.

[3] J. P. Sousa and D. Garlan, "Aura: an architectural framework for user mobility in ubiquitous computing environments," in *Software Architecture*. Springer, 2002, pp. 29–43.

[4] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*. ACM, 2002, pp. 87–92.

[5] S. Imai, T. Chestna, and C. A. Varela, "Elastic scalable cloud computing using application-level migration," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2012, pp. 91–98.

[6] V. Lee, H. Schneider, and R. Schell, *Mobile applications: architecture, design, and development*. Prentice Hall PTR, 2004.

[7] D. Kovachev, Y. Cao, and R. Klamma, "Mobile cloud computing: a comparison of application models," *arXiv preprint arXiv:1107.4940*, 2011.

[8] S. Blom, M. Book, V. Gruhn, R. Hrushchak, and A. Kohler, "Write once, run anywhere a survey of mobile runtime environments," in *Grid and Pervasive Computing Workshops, 2008. GPC Workshops' 08. The 3rd International Conference on*. IEEE, 2008, pp. 132–137.

[9] J. Jing, A. S. Helal, and A. Elmagarmid, "Client-server computing in mobile environments," *ACM computing surveys (CSUR)*, vol. 31, no. 2, pp. 117–157, 1999.

[10] X. Dai and J. Grundy, "Netpay: An off-line, decentralized micro-payment system for thin-client applications," *Electronic Commerce Research and Applications*, vol. 6, no. 1, pp. 91–101, 2007.

[11] J. Kim, R. A. Baratto, and J. Nieh, "pthinc: a thin-client architecture for mobile wireless web," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 143–152.

[12] 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*. ACM, 2010, pp. 49–62.

[13] 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*. ACM, 2011, pp. 301–314.

[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[15] M. L. Powell and B. P. Miller, *Process migration in DEMOS/MP*. ACM, 1983, vol. 17, no. 5.

[16] R. S. Gray, "Agent tcl: A flexible and secure mobile-agent system," 1997.

[17] B. D. Noble and M. Satyanarayanan, "Experience with adaptive mobile applications in odyssey," *Mobile Networks and Applications*, vol. 4, no. 4, pp. 245–254, 1999.

[18] P. Rong and M. Pedram, "Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach," in *Proceedings of the 40th annual Design Automation Conference*. ACM, 2003, pp. 906–911.

[19] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 795–809, 2004.

[20] S. Gurun, C. Krintz, and R. Wolski, "Nwslite: a light-weight prediction utility for mobile devices," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. ACM, 2004, pp. 2–11.

[21] S. Ou, K. Yang, A. Liotta, and L. Hu, "Performance analysis of offloading systems in mobile wireless environments," in *Communications, 2007. ICC'07. IEEE International Conference on*. IEEE, 2007, pp. 1821–1826.

[22] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2. IEEE, 2007, pp. 1–8.

[23] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *Communications Magazine, IEEE*, vol. 46, no. 1, pp. 56–63, 2008.

[24] 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.

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

[26] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, "Mobile cloud computing: A survey, state of art and future directions," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[27] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design and implementation of zap: A system for migrating computing environments," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 361–376, 2002.

[28] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution." in *HotOS*, vol. 9, 2009, pp. 8–11.

[29] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[30] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 127–134.

[31] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[32] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 53.

[33] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, "Availability and load balancing in cloud computing," in *International Conference on Computer and Software Modeling, Singapore*, vol. 14, 2011.

[34] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*. IEEE, 2010, pp. 89–96.

[35] P. Wang, W. Huang, and C. A. Varela, "Impact of virtual machine granularity on cloud computing workloads performance," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*. IEEE, 2010, pp. 393–400.

[36] R. Kaur and P. Luthra, "Load balancing in cloud computing," in *Int. Conf. on Recent Trends in Information, Telecommunication and Computing, ITC*, 2014.

[37] S. Bykov, A. Geller, G. Kliot, J. R. Larus, R. Pandya, and J. Thelin, "Orleans: cloud computing for everyone," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 16.

[38] V. Thornton, "Java object actors: An actor based service framework."

[39] M. Astley and G. A. Agha, "Customization and composition of distributed objects: Middleware abstractions for policy management," in *ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 6. ACM, 1998, pp. 1–9.

[40] M. Astley, D. C. Sturman, and G. Agha, "Customizable middleware for modular distributed software," *Communications of the ACM*, vol. 44, no. 5, pp. 99–107, 2001.

[41] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*. IEEE, 1999, pp. 2–10.

[42] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 168–178.

[43] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[44] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[45] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 335–348.

[46] M. B. Kjærgaard and H. Blunck, "Unsupervised power profiling for mobile devices," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2012, pp. 138–149.

[47] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "Devscope: a nonintrusive and online power analysis tool for smartphone hardware components," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2012, pp. 353–362.

[48] M. Kim, J. Kong, and S. W. Chung, "Enhancing online power estimation accuracy for smartphones," *Consumer Electronics, IEEE Transactions on*, vol. 58, no. 2, pp. 333–339, 2012.

[49] J. Lee, H. Joe, and H. Kim, "Automated power model generation method for smartphones," *Consumer Electronics, IEEE Transactions on*, vol. 60, no. 2, pp. 190–197, 2014.

[50] P. Moinzadeh, "I-admin: a framework for deriving adaptive service configuration in wireless smart sensor networks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2014.

[51] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 29–42.

[52] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring." in *USENIX Annual Technical Conference*, 2012, pp. 387–400.

[53] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, "Fepma: fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 367.

[54] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 85–90.

[55] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 735–737.

[56] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011.

[57] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 40–48, 1994.

[58] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[59] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," in *Foundations of Security Analysis and Design*. Springer, 2001, pp. 137–196.

[60] D. C. Latham, "Department of defense trusted computer system evaluation criteria," *Department of Defense*, 1986.

[61] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for xml documents," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 2, pp. 169–202, 2002.

[62] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," *Journal of Grid Computing*, vol. 7, no. 2, pp. 169–180, 2009.

[63] K. Z. Bijon, R. Krishnan, and R. Sandhu, "Towards an attribute based constraints specification language," in *Social Computing (SocialCom), 2013 International Conference on*. IEEE, 2013, pp. 108–113.

[64] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone et al., "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST Special Publication*, vol. 800, p. 162, 2013.

[65] J. Huang, D. M. Nicol, R. Bobba, and J. H. Huh, "A framework integrating attribute-based policies into role-based access control," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM, 2012, pp. 187–196.

[66] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First experiences using xacml for access control in distributed systems," in *Proceedings of the 2003 ACM workshop on XML security*. ACM, 2003, pp. 25–37.

[67] B. Cha, J. Seo, and J. Kim, "Design of attribute-based access control in cloud computing environment," in *Proceedings of the International Conference on IT Convergence and Security 2011*. Springer, 2012, pp. 41–50.

[68] Q. Ni and E. Bertino, "xfacl: an extensible functional language for access control," in *Proceedings of the 16th ACM symposium on Access control models and technologies*. ACM, 2011, pp. 61–72.

[69] A. Anderson, "A comparison of two privacy policy languages: Epal and xacml," 2005.

[70] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Computer Security–ESORICS 2009*. Springer, 2009, pp. 355–370.

[71] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," in *ACM SIGPLAN Notices*, vol. 39, no. 6. ACM, 2004, pp. 119–130.

[72] J. Liu, K. Kumar, and Y.-H. Lu, "Tradeoff between energy savings and privacy protection in computation offloading," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. ACM, 2010, pp. 213–218.

[73] C. Gentry, "Fully homomorphic encryption using ideal lattices." in *STOC*, vol. 9, 2009, pp. 169–178.

[74] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[75] P.-H. Chang and G. Agha, "Towards context-aware web applications," in *DAIS*, vol. 4531. Springer, 2007, pp. 239–252.

[76] P.-H. Chang and G. Agha, "Supporting reconfigurable object distribution for customized web applications," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 1286–1292.

[77] N. Venkatasubramanian, C. Talcott, and G. A. Agha, "A formal model for reasoning about adaptive qos-enabled middleware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 13, no. 1, pp. 86–147, 2004.

[78] M. R. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a pki environment," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 4, pp. 566–588, 2003.

[79] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellAgnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro, "Voms, an authorization system for virtual organizations," in *Grid computing*. Springer, 2004, pp. 33–40.

[80] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode, and K. Keahey, "Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy," in *5th Annual PKI R&D Workshop*, 2006.

[81] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 89–98.

[82] Z. Wan, J. Liu, and R. H. Deng, "Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 2, pp. 743–754, 2012.

[83] S. Ruj, A. Nayak, and I. Stojmenovic, "A security architecture for data aggregation and access control in smart grids," *arXiv preprint arXiv:1111.2619*, 2011.

[84] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology–ICISC 2009*. Springer, 2010, pp. 229–244.

[85] P. McDaniel and A. Prakash, "Methods and limitations of security policy reconciliation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 3, pp. 259–291, 2006.

[86] T. Yu and M. Winslett, "A unified scheme for resource protection in automated trust negotiation," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2003, pp. 110–122.

[87] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 14–22, 2009.

[88] F. J. Krautheim, "Private virtual infrastructure for cloud computing," in *Proceedings of the 2009 conference on Hot topics in cloud computing*. USENIX Association, 2009, pp. 5–5.

[89] R. K. Grewal and P. K. Pateriya, "A rule-based approach for effective resource provisioning in hybrid cloud environment," in *New Paradigms in Internet Computing*. Springer, 2013, pp. 41–57.

[90] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, 2006.

[91] G. Agha and C. Hewitt, "Concurrent programming using actors: Exploiting large-scale parallelism," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1985, pp. 19–41.

[92] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott, "A foundation for actor computation," *Journal of Functional Programming*, vol. 7, no. 01, pp. 1–72, 1997.

[93] W. Jansen, T. Grance et al., "Guidelines on security and privacy in public cloud computing," *NIST special publication*, vol. 800, p. 144, 2011.

[94] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 241–299, 2000.

[95] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Cloud Computing*. Springer, 2009, pp. 254–265.

[96] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," *Submitted to IFIP Networking*, 2014.

[97] B.-G. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 7.

[98] J. Armstrong, R. Virding, C. Wikström, and M. Williams, "Concurrent programming in erlang," 1993.

[99] J. Armstrong, *Programming Erlang: software for a concurrent world*. Pragmatic Bookshelf, 2007.

[100] M. Astley and T. Clausen, "Actor foundry," 2000.

[101] C. Varela and G. Agha, "Programming dynamically reconfigurable open systems with salsa," *ACM SIGPLAN Notices*, vol. 36, no. 12, pp. 20–34, 2001.

[102] P. Haller and F. Sommers, *Actors in Scala*. Artima Incorporation, 2012.

[103] M. Odersky, L. Spoon, and B. Venners, *Programming in scala*. Artima Inc, 2008.

[104] M. Gupta, *Akka Essentials*. Packt Publishing Ltd, 2012.

[105] R. K. Karmani, A. Shali, and G. Agha, "Actor frameworks for the jvm platform: a comparative analysis," in *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*. ACM, 2009, pp. 11–20.

[106] Wikipedia, "Eight queens puzzle," 2014, online; accessed 19-July-2014. [Online]. Available: http://www.wikipedia.org/wiki/Eight_queens_puzzle

[107] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 78–89.

[108] D. Kim, W. Jung, and H. Cha, "Runtime power estimation of mobile amoled displays," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 61–64.

[109] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: building secure cloud framework for mobile computing and communication," in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*. IEEE, 2010, pp. 27–34.

[110] R. Chow, M. Jakobsson, R. Masuoka, J. Molina, Y. Niu, E. Shi, and Z. Song, "Authentication in the clouds: a framework and its application to mobile users," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*. ACM, 2010, pp. 1–6.

[111] X. Yu and Q. Wen, "Design of security solution to mobile cloud storage," in *Knowledge Discovery and Data Mining*. Springer, 2012, pp. 255–263.

[112] A. N. Khan, M. Mat Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1278–1299, 2013.