Mobile Cloud Application Framework and Offloading Strategies

by

Huijun Wu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2016 by the
Graduate Supervisory Committee:

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

Mobile Cloud computing has shown its capability to support mobile devices for provisioning computing, storage and communication resources. A distributed mobile cloud service system called “POEM” is presented to manage the mobile cloud resource and compose mobile cloud applications. POEM considers resource management not only between mobile devices and clouds, but also among mobile devices. It implements both computation offloading and service composition features. The proposed POEM solution is demonstrated by using OSGi and XMPP techniques.

Offloading is one major type of collaborations between mobile device and cloud to achieve less execution time and less energy consumption. Offloading decisions for mobile cloud collaboration involve many decision factors. One of important decision factors is the network unavailability. This report presents an offloading decision model that takes network unavailability into consideration. The application execution time and energy consumption in both ideal network and network with some unavailability are analyzed. Based on the presented theoretical model, an application partition algorithm and a decision module are presented to produce an offloading decision that is resistant to network unavailability.

Existing offloading models mainly focus on the one-to-one offloading relation. To address the multi-factor and multi-site offloading mobile cloud application scenarios, a multi-factor multi-site risk-based offloading model is presented, which abstracts the offloading impact factors as for offloading benefit and offloading risk. The offloading decision is made based on a comprehensive offloading risk evaluation. This presented model is generic and expendable. Four offloading impact factors are presented to show the construction and operation of the presented offloading model, which can be easily extended to incorporate more factors to make offloading decision more comprehensive. The overall offloading benefits and risks are aggregated based on the mobile cloud users' preference.

The offloading topology may change during the whole application life. A set of algorithms are presented to address the service topology reconfiguration problem in several mobile cloud representative application scenarios, i.e., they are modeled as finite horizon scenarios, infinite horizon scenarios, and large state space scenarios to represent ad hoc, long-term, and large-scale mobile cloud service composition scenarios, respectively.

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. Dijiang Huang, for his consistent support during my Ph.D study. Professor Dijiang Huang helped to cultivate my research strength, provide constant invaluable advices in the last five years. Together with him, I learned to write my first paper and accomplish many challenging problems. More importantly, I learned from him the work attitude, ethic and disciplines that I will benefit for a life long. I feel very lucky to have him as my advisor, a teacher, and a life long friend.

I would like to thank my dissertation committee, Guoliang Xue, Partha Dasgupta, and Pitu Mirchandani, for their help, encouragements, and the constructive feedback that they have provided. I have a diverse committee and they have all been an inspiration for my research. In particular, I would like to thank Dr. Mirchandani for the support on the MIDAS project.

Members of our Secure Networking And Computing (SNAC) lab inspired me a lot through discussions, seminars, and project collaborations, and I would like to thank the following people for their valuable interactions: Chun-Jen Chung, Yuli Deng, Abdullah Alshalan, Ankur Chowdhary, Sandeep Pisharody, Duo Lu, Fanjie Lin, Weijia Wang, Bing Li, Zhijie Wang, Pankaj Kumar Khatkar, Tianyi Xing, Le Xu, Xinyi Dong, Yunji Zhong, Yang Qin, Bo Li, Zhiyuan Ma, Jingsong Cui.

Over the years, many friends and colleagues have provided me very helpful suggestions, insightful comments, support and encouragement during my Ph.D journey. I would like to thank Yiming Jing, Guohua Hu, Yang Cao, Jingchao Sun, Shaobo Zhang, Xiaowen Gong, Mengyuan Zhang, Kerem Demirtas, Jane Perera, Wei Zhou, Ahmet Altay.

Last but not the least, I would like to thank my family for the support that they provided through out my entire life and graduate school.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Mobile Cloud Computing (MCC) is the combination of Mobile Computing (MC), Cloud Computing (CC) and wireless networks to bring rich computational resources to mobile users, network operators, as well as cloud computing providers. More comprehensively, MCC can be defined as a rich mobile computing technology that leverages unified elastic resources of clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms.

The differences of MCC from MC and CC can be clarified in the following aspects: 1) Involved parties: The mobile devices and clouds are mandatory components for MCC. MC involves mainly mobile devices, however, clouds are not mandatory. CC involves mainly clouds, however, the mobile devices are not mandatory. 2) Computation and storage resource: The mobile devices usually have limited computation and storage resources, thus MC applications have to consider resource consumption limitation. The CC has much more resources and the resources can be added or removed dynamically. MCC enables the mobile devices to leverage the cloud resources, thus MCC like CC can handle resource intensive applications. 3) Network communication: The MC network communication include ad-hoc and infrastructure networks as the involved mobile devices can form flexible and complex wireless communication scenarios. The CC network communication involves data center networking and software defined networking. MCC network communication bridges the wireless MC network and the virtualized CC network, which connects mobile devices and servers in the clouds. 4) Mobility and availability: The mobility is one of the key properties for MC. The mobile devices may go outside of the radio area and be offline. CC does not have mobility property itself, and the clouds are supposed to provide services all the time. MCC like MC has to consider mobility. For example, the application tries to offload the computation to the clouds when the connection to cloud is available and it has to handle the computation when the device goes out of radio area. 5) User interaction: MC provides the direct user-device interaction by the user-friendly apps on the mobile devices. CC usually provides the services to the end user indirectly through the near user devices or softwares, such as smart phones or web browsers. MCC like MC is close to the end users and bridges the CC to the end users. 6) Tenant and sharing: Multiple users share the resources by tenants in CC. The mobile devices do not have tenants and usually not share the hardware resources in MC. For MCC, the mobile devices connect to one or multiple tenants to consume the shared cloud resources.

1.1 Mobile Cloud Running System Challenges

In mobile clouds, mobile devices and cloud resources compose a distributed mobile application running environment, where a mobile application may consume resources from both local and remote resource providers who provide computing, networking, sensing, and storage resource provisioning. Mobile devices can serve as either service consumers or service providers in the mobile cloud, in which the cloud boundaries are extended into the mobile domain [Huang *et al.*, 2013a][Huang *et al.*, 2011]. Mobile applications may require a mobile device to interact with other mobile devices and cloud resource providers to achieve desired computing, storing or sensing features.

An ideal mobile cloud application running system should enable mobile devices to easily discover and compose mobile cloud resources for its applications. From mobile resource providers' perspectives, they may not even know what applications are using their resources and who may call their provisioned functions beforehand. In this way, the mobile application design should not be application-oriented; instead, it should be functionality-oriented or service-oriented. For example, the video function of a mobile device should provide general interfaces that can be called by multiple local or remote functions in the runtime. To achieve this feature, we can consider these Provisioning Functions (PFs) as the fundamental application components in the mobile cloud, which can be composed by mobile cloud service requesters in the runtime. As a result, mobile cloud can significantly reduce the mobile application development overhead and greatly improve the agility and flexibility to build a personalized mobile cloud computing system that can be customized for each mobile user.

A simple collaborative vehicular video sensing example is used to illustrate the above described mobile cloud features in Figure 1.1. Alice is driving on the road and her smartphone, which is mounted on the front dashboard for navigation, has the basic video capture PF. Bob is driving after Alice and is running an image processing PF in his phone and wants to utilize the video clips from the vehicles before him in order to reconstruct the road situations ahead. Then Bob can consume Alice's video PF to reconstruct the view of the road segment. Moreover, Alice and Bob cab share the captured video clips to their friend Carol who is managing a traffic monitoring website that posts videos from smartphone users for the public to access the realtime road traffic information and provides the optimal route suggestions. In this mobile application scenario, all participants have their basic PFs: (a) Alice: video capture, (b) Bob: video process and augment, and (c) Carol: video display and route suggestion. Note that a PF can be called by multiple other PFs for different application purposes, and they altogether can build several mobile cloud applications.

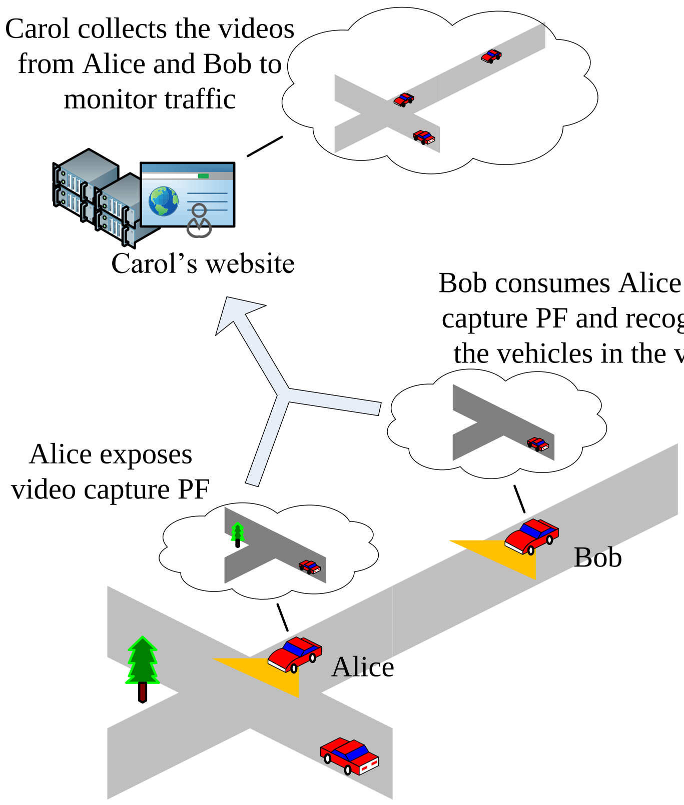

Carol collects the videos from Alice and Bob to monitor traffic
Carol's website
Bob consumes Alice video capture PF and recognizes the vehicles in the video
Alice exposes video capture PF
Bob
Alice

Figure 1.1: Example: Collaborative Vehicular Video Sensing.

There are several challenges invoked by the above described application scenario. The first challenge is that knowing the status of mobile devices, e.g., online/offline and runtime information (such as battery, computing power, connectivity, etc.), is difficult due to the mobility of mobile users. The second challenge is that knowing the available PFs on each mobile device is not a trivial task. Currently, there is no such a common framework allowing mobile devices for exchanging the available PFs information and running such a system in a distributed environment. The third challenge is to compose PFs crossing various hardware and software platforms, which demands a universal programming and application running environment with little compatibility issues. A mobile cloud application running system to solve these challenges is presented in the Chapter 3.

1.2 Offloading Strategy Challenges

The computation power and device battery are limited due to the device size and weight. The computation offloading is the most important topic in mobile cloud computing because offloading may not only save mobile device energy but also improve the application performance. However, offloading is not the final solution. Offloading introduces cost on bandwidth and energy as well as risk on the application availability. We should consider the offloading strategies carefully to get the best benefit.

1.2.1 Network Unavailability

The collaboration of mobile device and cloud enlarges the advantages of both mobile device and cloud. The mobile device can use the unlimited computation and storage resource of cloud, and meanwhile, the cloud gets more close to end user through the bridge of mobile device. Mobile cloud application based on mobile cloud collaboration deploys its components into different places including local smart phone and virtual machines in cloud. The application initially starts all components on smart phone locally. When remote cloud resource becomes available, the mobile cloud application may offload computing-intensive or memory-consuming components to remote cloud to improve performance or save mobile device energy. One key issue in offloading process is how to make offloading decisions. Due to the complexity of mobile and cloud environments, offloading decision involves mobile side parameters such as CPU and memory, cloud side environment including virtual machine capability and performance, and the state of network in between, which altogether makes the decision making problem difficult to tackle. The network state is the most complicated and important part among three aspects considered.

Network status has great impact on mobile cloud collaboration. Due to mobility of mobile users, the network connecting mobile device and cloud changes drastically. The network connection may be lost when mobile device moves into some area that is not covered by wireless network. When the connection to cloud is lost, the original execution routine is interrupted and the application has to wait until the connection resumes [Flinn, 2012]. The expected benefit of mobile cloud collaboration may not be obtained due to interruption of execution plan. The collaboration may even lead to negative impact to execution time or energy consumption in such scenario. Thus, we need to find offloading solution that is resistant to network unavailability to assure the less execution time and less energy consumption benefits.

1.2.2 Multi-Objective

In mobile cloud computing, offloading is an important approach to overcome the resources and functionalities constrains of mobile devices. In a mobile cloud offloading model, applications deploy their components on multiple application processing nodes such as mobile smart phones and Virtual Machines (VMs) in a cloud. A mobile device can rely on an offloading decision model based on multiple factors such as offloading computational-intensive, time- or energy-consuming application functions. Moreover, the offloading model should allow one-to-many (i.e., multi-site) offloading to improve the flexibility of the construction the running application. Furthermore, the mobile device should consider risk issues such as privacy preservation and reliability of offloading targets, etc. A general probabilistic framework to model this multi-factor, multi-site, risk-based mobile cloud application offloading is in need.

To illustrate the mobile cloud multi-factor, multi-site application offloading scenarios, we present an illustrative vehicular traffic monitoring example in Figure 1.2. In this example, a realtime traffic monitoring application running on a mobile device

A wants to combine video views from other vehicles B, C and D (e.g., through on-board cameras or smart phones, etc.) to get a panoramic view of a road intersection that the user will pass. Assume that there are cameras on the vehicles and every vehicle shares its captured videos or pictures to other vehicles through a location-based social network model (e.g., Waze, Foursquare). In the system, each mobile node has a dedicated supporting node, e.g., a VM in the cloud, where computation and trust management operations can be offloaded to VMs. This mobile cloud application follows the user-centric mobile cloud computing model recently presented in [Huang *et al.*, 2013b], in which the videos captured by vehicles can be uploaded to their VMs in cloud for video data processing. During the runtime, the application first retrieves the views from VMs of vehicles located on the requested road intersection. The application processes the collected videos in the cloud to generate a panoramic view at a given time, and then get back to the user. In this application scenario, multiple offloading targets such as VMs and vehicles exit; application components (ACs), such as video clip capturing, processing (locally on the mobile device), and panoramic view generation can be offloaded to VMs and vehicles for energy saving and reducing the delay due to video data collection and panoramic view generation.

Many existing mobile cloud offloading models focus on a one-to-one directional offloading model (e.g., [Cuervo *et al.*, 2010][Kosta *et al.*, 2012][Zhang *et al.*, 2011]): i.e., offloading from a mobile device to a cloud server (e.g., a VM). Studies in [Ou *et al.*, 2006] and [Sinha and Kulkarni, 2011] had shown the benefits of using one-to-many (i.e., multi-site) offloading models to improve the performance of mobile devices, where a mobile device can offload multiple application functions to multiple computing nodes. However, existing offloading models only consider one offloading factor such as computation overhead, networking delay, or energy consumption. Moreover, they do not consider other important offloading factors such privacy, reliability, etc.

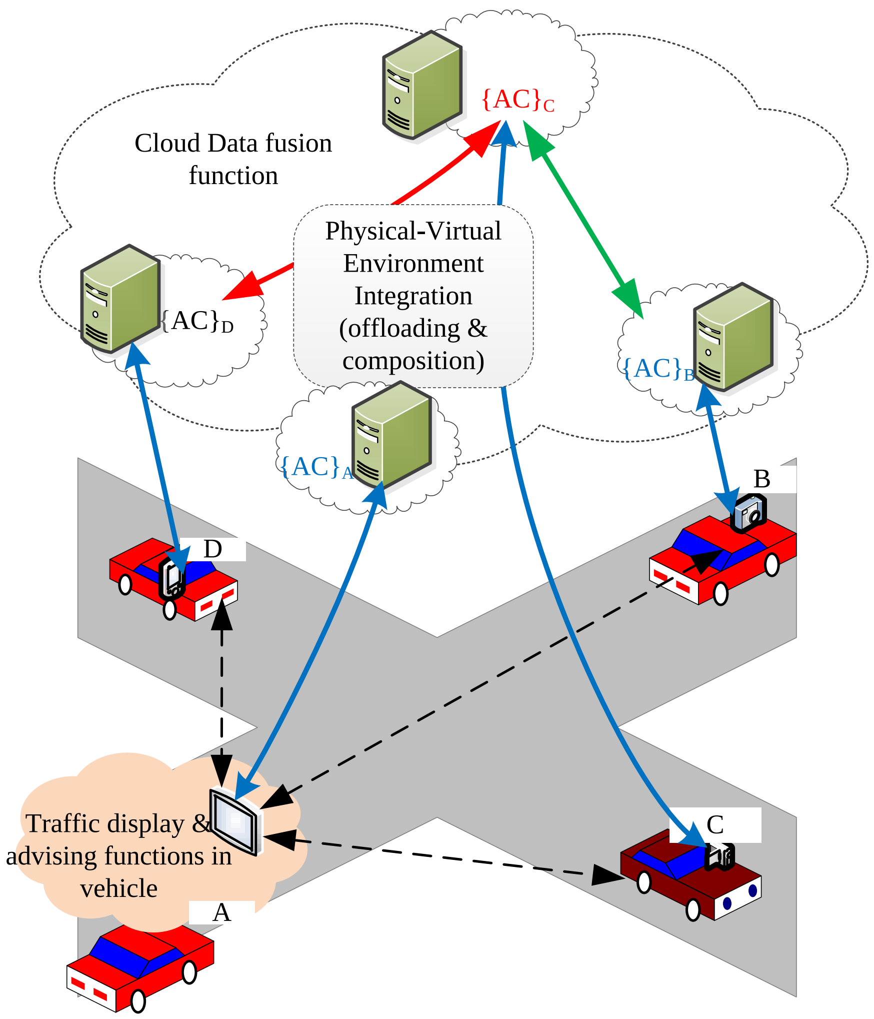

Cloud Data fusion function
{AC}$_C$
Physical-Virtual Environment Integration (offloading & composition)
{AC}$_D$
{AC}$_B$
{AC}$_A$
B
D
C
A
Traffic display & advising functions in vehicle

Figure 1.2: Example: Constructing Panoramic View of a Road Intersection.

One of important focuses of this work is to model the offloading risks that have not been addressed by previous research. Particularly, we focus on two main sources of risks in mobile cloud computing: privacy breach and offloading reliability. Mobile application components can be offloaded to malicious nodes that can potentially compromise the application's privacy, e.g., the offloaded data and functions may contain private data or expose the purpose of the application. In addition to the privacy-breach risk, another offloading risk is the reliability of offloading targets. For example, the surrogates may be unavailable due to unstable communication link or software crash, etc. In this case, offloading effort has to include offloading recovery strategies.

Thus, both privacy breach and reliability risks should be considered in the offloading model in addition to the performance offloading factors described previously.

1.2.3 Offloading Series

Mobile cloud applications usually incorporate resources from mobile devices and virtual machines in clouds [Huang *et al.*, 2013a]. Prior work of mobile cloud computing [Kemp *et al.*, 2012][Kosta *et al.*, 2012][Cuervo *et al.*, 2010][Chun *et al.*, 2011] mainly focuses on a simple one-to-one service offloading scenario: i.e., a mobile device offloads its services to a cloud.

Nowadays, many mobile cloud applications adopt a multi-site service model [Sinha and Kulkarni, 2011][Ou *et al.*, 2006][Liu, 2013] to maximize the benefits from multiple service providers, where the research focuses on how to compose services that have been already implemented on multiple service providers. We do not differentiate application functions and services since our model can be applied for both application function calling models or service oriented architecture. A simple example can be used to highlight the application scenario: a mobile device calls a video capturing function from multiple remote mobile devices at a congested road segment, an image processing function to recognize vehicles on the road segment from the cloud, and then the requesting mobile device uses its local UI to display the road traffic with identified lanes and vehicles. Compared to traditional approach that users upload captured videos to the cloud and the requester downloads the processed videos from the cloud, the presented application scenario does not require a pre-setup application scenario to each function. This approach is very flexible in terms of ad hoc application composition, and it can maximally improve the resource utilization such as the video capturing function of mobile devices can be shared by multiple users for different purposes, e.g., congestion monitoring, road events detection , etc.

Besides the flexibility of the service/function composition using multi-site service composition, it also introduces benefits in reducing execution time and energy consumption of mobile devices; however, at the same time, it brings more management issues as well. The multi-site service composition paradigm involves multiple surrogate sites in the service composition process, and the application running environment changes demand a decision model in the service composition decision processes to consider the application running environment changes related to network connectivity, delay, device resource consumption, energy usage, etc. Moreover, due to the mobility, the benefits of service composition may not be consistent during the entire application execution time period. To adapt to the application running environment changes, we need to study the service reconfiguration strategy by modeling the service composition as a service-topology mapping (or reconfiguration) problem that is illustrated in Fig. 1.3. Relying on a cloud service framework, once the mobile application running environment state changes, a new service-topology reconfiguration is decided, and then the application components are redistributed to the surrogate sites through the cloud. In this way, the service composition is adaptive to gain the maximum service benefits during the entire execution time period.

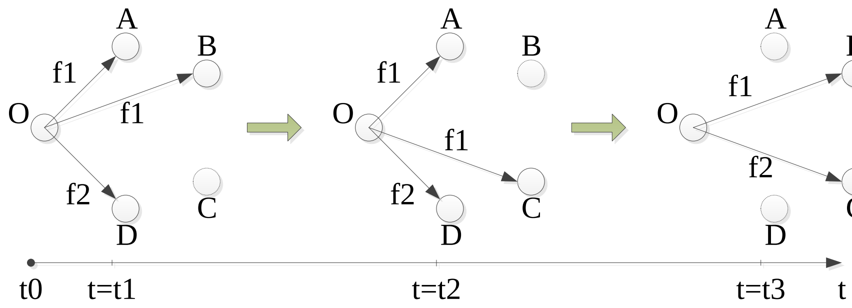


Figure 1.3: Service Composition Topology Reconfiguration Motivation.

The previous vehicle-based image capturing and processing example is presented in Fig. 1.3, where in the crowd vehicle collaborative application scenario, the requestor's function O calls an image capturing function $f1$ from collocated devices A and B at time $t1$ to derive a panoramic view on their location. Vehicle B may move out of service region at time $t2$, and then O calls vehicle C to replace B when C moves into the service location. From $t1$ to $t2$ and $t2$ to $t3$, the service topology reconfiguration takes place and it as well improves the system reliability since the task may be reconfigured to other available surrogate sites providing redundancy in case one of them fails.

To solve the challenges in Section 1.1 and Section 1.2, I propose a mobile cloud running system in Chapter 3 and the offloading strategies in Chapter 4. An application example of proposed work is described in Chapter 5.

Chapter 2

RELATED WORK

The recent mobile cloud research can be categorized into three areas: mobile cloud systems, mobile cloud offloading strategies, and mobile cloud applications such as vehicular cloud.

2.1 Mobile Cloud System

Most of the research on the mobile cloud computing system solutions fall into two categories by used technologies: service oriented-based approach or virtualization. The following proposed solutions use service oriented-based approach. SCAMPI [Pitkänen *et al.*, 2012] is an architecture proposed to support distributed task execution in opportunistic pervasive networks. The key elements of the architecture include leveraging human social behavior for efficient opportunistic interaction between a variety of sensors, personal communication devices and resources embedded in the local environment. The SCAMPI architecture abstracts resources as service components following a service-oriented model. MAPCloud [Rahimi *et al.*, 2012] is a hybrid, tiered cloud architecture consisting of local and public clouds, which can be leveraged to increase both performance and scalability of mobile applications. It models the mobile application as a workow of tasks and aim to optimally decompose the set of tasks to execute on the mobile client and 2-tier cloud architecture considering multiple QoS factors such as power, price, and delay. Cuckoo [Kemp *et al.*, 2012] is a practical implementation of computation offloading for Android, which simplifies the development of smartphone applications that benefit from computation offloading and provides a dynamic runtime system, that can, at runtime, decide whether a part of an application will be executed locally or remotely. Carmen [Kim *et al.*, 2012] is a distributed system that manages the mobile connectivity of a set of devices belonging to a particular individual, called the Mobile Personal Grid (MPG). Carmen enables the MPG to efficiently collect context from a mobile user and coordinate key system resources across the MPG and cloud. μCloud [March *et al.*, 2011] focuses on two main issues in cloud-enabled mobile applications, namely complexity of application development and offline usability. μCloud framework models a rich mobile application as a graph of components distributed onto mobile devices and the cloud. Scavenger [Kristensen, 2010] is a cyber foraging system supporting easy development of mobile cyber foraging applications, while still delivering efficient, mobile use of remote computing resources through the use of a custom built mobile code execution environment and a new dual-profiling scheduler. Zhang et al. [Zhang *et al.*, 2010b] propose a model that is based on elastic applications technique, where a single elastic application is partitioned into multiple components called weblets. A weblet can be defined as an independent functional unit of an application that can compute, store, and communicate while keeping its execution location transparent. Huerta-Canepa et al. [Huerta-Canepa and Lee, 2010] create a virtual cloud computing platform using mobile phones to overcome the cloud access issue since an access to the cloud platforms is not always guaranteed to be available and/or it is too expensive to access them. VOLARE [Papakos *et al.*, 2010] is a middleware-based solution that monitors the resources and context of the device, and dynamically adapts cloud service requests accordingly, at discovery time or at runtime.

The following proposed solutions use virtualization technology. Thinkair [Kosta *et al.*, 2012] exploits the concept of smartphone virtualization in the cloud and provides method-level computation offloading. It focuses on the elasticity and scalability of the cloud and enhances the power of mobile cloud computing by parallelizing method execution using multiple virtual machine images. CloneCloud [Chun *et al.*, 2011] automatically transforms mobile applications to benefit from the cloud. The system is a flexible application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. eXCloud [Ma *et al.*, 2011] is a middleware system with multi-level mobility support, ranging from as coarse as a VM instance to as fine as a runtime stack frame, and allows resources to be integrated and used dynamically. In eXCloud, a Stack-On-Demand (SOD) [Ma *et al.*, 2010] approach is used to support computation mobility throughout the mobile cloud environment. MAUI [Cuervo *et al.*, 2010] enables fine-grained energy-aware offload of mobile code to the infrastructure. it supports fine-grained code offload to maximize energy savings with minimal burden on the programmer.

2.2 Offloading Strategies

The mobile cloud computing systems partition the applications to distribute execution and/or data to multiple nodes. There are several state-of-the-art partition algorithms. Yang et al. [Yang *et al.*, 2013] studied the partitioning problem for mobile data stream applications, where the optimization is placed on achieving high throughput of processing the streaming data rather than minimizing the makespan of executions as in other applications. They designed a genetic algorithm for optimal computation partition. Abebe et al. [Abebe and Ryan, 2011] proposed a type of adaptation granularity which combines the efficiency of coarse level approaches with the efficacy of fine-grained adaptation. An approach for achieving this level of granularity through the dynamic decomposition of runtime class graphs was presented. Abebe et al. [Abebe and Ryan, 2012] presented a distributed approach to application representation in which each device maintains a graph consisting only of components in its memory space, while maintaining abstraction elements for components in remote devices. An extension to an existing application graph partitioning heuristic is proposed to utilize this representation approach. Giurgiu et al. [Giurgiu *et al.*, 2012] developed a system that dynamically adapts the application partition decisions. The system works by continuously profiling an applications performance and dynamically updating its distributed deployment to accommodate changes in the network bandwidth, devices CPU utilization, and data loads. Sinha et al. [Sinha and Kulkarni, 2011] described algorithmic approaches for performing fine-grained, multi-site offloading. This allows portions of an application to be offloaded in a data-centric manner, even if that data exists at multiple sites. Kovachev [Kovachev, 2012] presented Mobile Augmentation Cloud Services (MACS) middleware which enables adaptive extension of Android application execution from a mobile client into the cloud. MACS uses a dynamic partitioning scheme, and lightweight as extra profiling. Resource monitoring is performed for adaptive partitioning decision during runtime Ra et al. [Ra *et al.*, 2012] experimentally and analytically investigate the design considerations - which segments of the application are most efficient to be hosted on the low power processor, and how to select an appropriate low power processor. Linear programming was applied. Smit et al. [Smit *et al.*, 2012] described an approach to partitioning a software application into components that can be run in the public cloud and components that should remain in the private data center. Static code analysis is used to automatically establish a partitioning based on low-effort input from the developer. Niu et al. [Niu *et al.*, 2014] took the bandwidth as a variable to improve static partitioning and avoid high costs of dynamic partitioning. They proposed the Branch-and-Bound based Application Partitioning (BBAP) algorithm and Min-Cut based Greedy Application Partitioning (MCGAP) algorithm based on application

Object Relation Graphs (ORGs) by combining static analysis and dynamic profiling. Verbelen et al. [Verbelen *et al.*, 2013] designed graph partitioning algorithms that allocate software components to machines in the cloud while minimizing the required bandwidth. Their algorithms are not restricted to balanced partitions and take into account infrastructure heterogeneity. Besides the above work, [Chen *et al.*, 2015][Chakareski, 2013][Corradi *et al.*, 2014][Gerla, 2012][Hsu *et al.*, 2014] discussed the mobile cloud system in specific areas.

2.3 Vehicular Cloud

Vehicular cloud, as an typical example of mobile cloud, earns more and more attention. The recent research on vehicular cloud falls into three categories according to cloud service layers: communication, resource scheduling, and application. The vehicular communication is usually based on current communication techniques, such as 3 G or 4 G cellular communication devices, Wi-Fi, WiMAx, Wireless Access in Vehicular Environment (WAVE) [Jiang and Delgrossi, 2008], or Dedicated Short Range Communication (DSRC) [Xu *et al.*, 2003]. Based on these techniques, the Vehicular-to-Vehicular (V2V) communication [Yang *et al.*, 2004] and Vehicle-to-Infrastructure (V2I) are established.

Since the storage size becomes smaller and the price goes cheaper, the storage in the vehicle on-board computer has much storage space. The vehicles with additional storage capability can provide its storage space as a service. Meanwhile, the compute power becomes stronger, which leads to compute as service [Olariu *et al.*, 2013]. Arif et al. [Arif *et al.*, 2012] envision a vehicular cloud involving cars in the long-term parking lot of a typical international airport. The patrons of such a parking lot are typically on travel for several days, providing a pool of cars that can serve as the basis for a datacenter at the airport. The authors provide closed forms for the probability distribution of the parking lot occupancy as a function of time, for the expected number of cars in the parking lot and its variance, and for the limiting behavior of these parameters as time increases.

Eltoweissy et al. [Eltoweissy *et al.*, 2010] proposed a novel and more comprehensive vision namely, that advances in vehicular networks, embedded devices, and cloud computing will enable the formation of autonomous clouds of vehicular computing, communication, sensing, power and physical resources. Hence, the authors coin the term, Autonomous Vehicular Clouds (AVCs). A key features distinguishing AVCs from conventional cloud computing is that mobile AVC resources can be pooled dynamically to serve authorized users and to enable autonomy in real-time service sharing and management on terrestrial, aerial, or aquatic pathways or theatres of operations. NAVOPT [Kim and Gerla, 2011] is a vehicular routing strategy assisted by the on board navigator as well as the navigation server. The on board navigator equipped with area map and GPS monitor, reports its own position to the server via wireless connection (WiFi or 3G), in turns, it acquires from the Server a minimum cost path (i.e. path with shortest travel time) under the current traffic conditions. Under NAVOPT, the Server uses a Flow Deviation (FD) algorithm to compute optimal vehicle routes by load balancing vehicle traffic over alternate routes. Alazawi et al. [Alazawi *et al.*, 2011] leverage Intelligent Transportation Systems (ITS) including Vehicular Ad hoc Networks (VANETs), mobile and Cloud computing technologies to propose an intelligent disaster management system. The system is intelligent because it is able to gather information from multiple sources and locations, including from the point of incident, and is able to make effective strategies and decisions, and propagate the information to vehicles and other nodes in real-time.

Chapter 3

MOBILE CLOUD APPLICATION FRAMEWORK

A new mobile cloud application running system is presented in this chapter, which is called POEM (Personal On-demand execution Environment for Mobile cloud computing), as shown in Figure 3.1. POEM incorporates both Provisioning Function (PF) offloading and PF composition features. POEM treats each mobile device as a PF provider. In addition, POEM is designed based on the mobile cloud framework, where a dedicated VM is assigned to each mobile device providing computing and storage support. Moreover, PFs can be offloaded/migrated from a mobile device to its assigned VM. The VM can not only run mobile devices' PFs (i.e., as shadows), but also can run extended PFs that mobile devices may not have the capacity to execute. Thus, we also call the VM in the POEM framework as Extended Semi-Shadow Image (ESSI) [Huang *et al.*, 2011]. In addition to offloading PF to ESSI, a mobile device can also offload PFs to friends' mobile devices, which makes computation close to the source data and reduces the communication cost in some circumstances. Thus, both mobile devices and ESSI can act as surrogates. Collectively, the PFs provided by a mobile device X and its corresponding $ESSI_X$ is denoted as $\{PF\}_X$. POEM regards both mobile devices and their dedicated ESSIs as PF providers. As a result, the mobile user's applications can be composed by PFs from local PFs (may be offloaded/migrated to its dedicated ESSI) and/or remote PFs (may run on remote mobile devices or their dedicated ESSIs). PF composition makes it easy to implement personalized application by composing the modularized PF. The PF offloading keeps data location and migrates computation, while the PF composition keeps computation location and migrates data.

To demonstrate the proposed POEM solutions, a pilot POEM system is implemented based on OSGi [OSGi-Alliance, 2012] and XMPP [Saint-Andre, 2011] techniques. In summary, the contributions of the POEM system is highlighted as follows:

- *Versatile and personalized application offloading, migration, and composition:* POEM maintains available mobile cloud resource and allows users choosing a mobile cloud application by using different approaches (offloading, migration, and composition) based on the available system resources and their personalized application requirements.

- *Social mobile cloud computing:* POEM solution enables mobile cloud application to utilize social network power, i.e., in addition to the discovered PFs through the mobile cloud system, mobile user can establish mobile cloud applications through their trusted social connections. In this way, POEM applications not only can use the resource in cloud by offloading resource intensive components to ESSI but also can use services provided from their social connections.

- *On-demand and adaptability:* POEM monitors connectivity of mobile devices through the XMPP service model, and provides real-time PFs availability information and history service profiles of mobile devices. For non-hardware-dependent PFs, the dedicated ESSI can represent its associated mobile device, which provides a flexible and agile application running environment for POEM applications.

- *Reduced code intrusion:* POEM does not restrict application structure; additionally, it does not require programmers' specific instruction during the application composition phase. POEM applications gain flexibility and code reuse due to the introduced OSGi modularization.

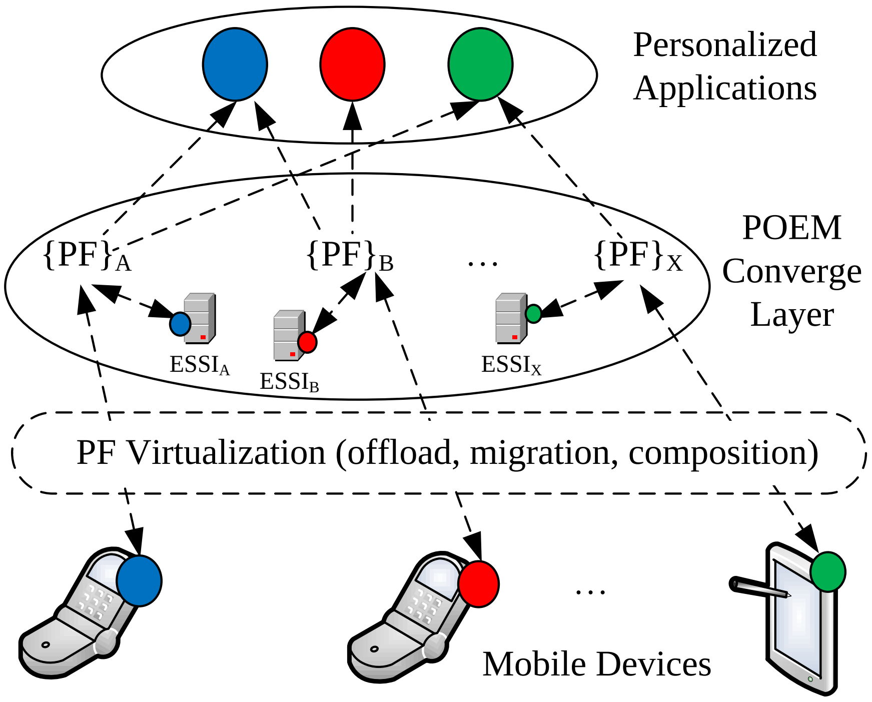


Figure 3.1: Overview of POEM system.

The chapter is organized as follows. Section 3.1 describes systems, application model and execution model. Section 3.2 discusses POEM architecture and design details. Section 3.3 discusses selected POEM implementation issues. Section 3.4 presents evaluation results including macro-benchmarks and micro-benchmarks.

3.1 System and Models

The POEM system is distributed framework. Each mobile devices or the VM in the cloud runs POEM framework instances to join the POEM system. The mobile users' applications, which consists of multiple PFs, run on top of the POEM framework. A PF is a piece of modularized code, which can be exposed as a service or migrated to other POEM frameworks. A PF may consume the services provided by other PFs locally or remotely. In short, the POEM framework is a container for PFs that work together to fulfill application tasks by service composition and offloading.

The POEM system is based on OSGi framework [OSGi-Alliance, 2012] that is a general purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications. Due to the popularity of Java, OSGi framework is compatible with major operating systems for both desktop and mobile device systems. The framework is stacked in layers: from bottom-up, module layer, life cycle layer, and service layer. The framework defines a unit of modularization, called a bundle, i.e., "PF" in the POEM. In the later descriptions, we do not differentiate the terms bundle and PF. In POEM, a PF is comprised of Java classes and other resources, and is deployed as a $\underline{\text{J}}$ava $\underline{\text{AR}}$chive (JAR) file. PF sits on the top of stacked layers and interacts with them through PF context. Module layer and life cycle layer handle PF installation and activation. PF can be installed/uninstalled and started/stopped. Service layer has a service registry and handles service publication and discovery. A service is a normal Java object that is registered under one or more Java interfaces with the service registry. PFs can register services, search for them, or receive notifications when services' states change. When PF is installed, the framework must cache the PF JAR file. A SERVICE_RANKING property may be specified when a service is being registered. The service with the highest ranking is returned when the framework receives service query. Before the service is consumed, it may become a stale reference. Service tracker is usually used for service consumer to prevent stale reference by obtaining reference when consumption happens. Besides local service activities, a distribution provider can export service to another framework by creating end point or import service from another framework by creating proxy for service composition, and then registering the proxy as an imported service. The rest of this section presents application and execution model of the proposed POEM system.

3.1.1 Application Model

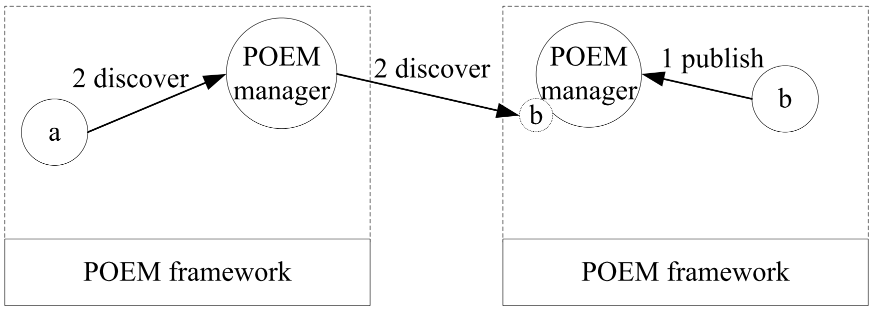
The POEM models a mobile cloud application as a set of PFs. The PF may provide class definitions and host services that implements PFs. POEM does not differentiate PFs on mobile devices and PFs on their ESSIs as the PFs may be migrated from mobile side to cloud side or vice versa without any modification. The uniform PF format make PFs reusable and reduces develops' workload by avoiding developing separate PFs for specific platform. The application may use services provided by local or remote PFs. The application can migrate PFs between mobile devices and ESSIs without disrupting the other active PFs.

POEM achieves social feature through an implemented XMPP [Saint-Andre, 2011] system within the MobiCloud system [Huang *et al.*, 2010]. The availability information of the system resources and mobile devices is maintained through a decentralized client-server architecture, where every mobile cloud entity needs an address called a JabberID (JID). JID is presented in the form of user@domain/resource, where 'domain' represents the XMPP service provider, 'user' represents virtual identity in the domain, and 'resource' identifies connection to an XMPP server. Three basic status services are achieved through XMPP: message, presence, and info/query (i/q). POEM's service discovery protocol provides two discovery methods: one enables discovering information about an entity; and the other enables to discover the items associated with an entity. In POEM, each entity, i.e., a mobile device or an ESSI, runs an OSGi framework, which is identified uniquely by its JID. One POEM entity discovers services hosted by his/her friends through XMPP service discovery protocol and XMPP publish-subscribe protocol. Mobile applications offload PFs to ESSIs through XMPP file transfer protocol, and the data exchange with remote application in POEM is through XMPP i/q communication.

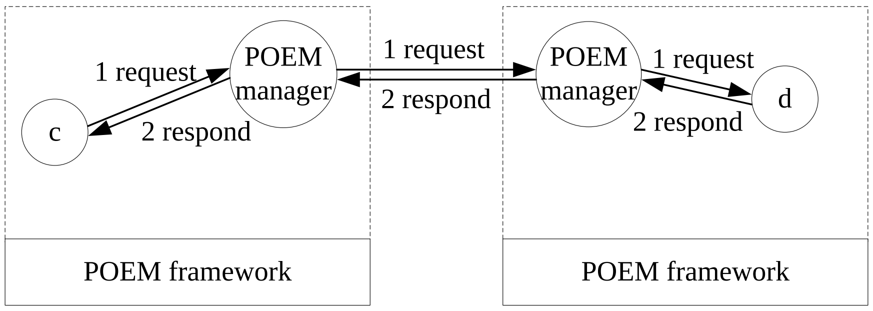
Reviewing the motivation example in Figure 1.1, two application models can be summarized:

- First, the PFs can work together through service binding. One PF can expose the service that is supposed to be consumed by other PFs. The service binding composes the simple services from each PF to build a complex or complete application. The service binding is a simple but powerful method to build a complex application. PFs exchange data to feed the service and get the result. The PFs themselves do not have to know when or where input data will come, but simply accept requests and process them. Although the service binding can handle complex application tasks, but it may not be efficient. In some scenarios, the data amount is quite large while the data processing is quite fast so that data exchange cost is high.

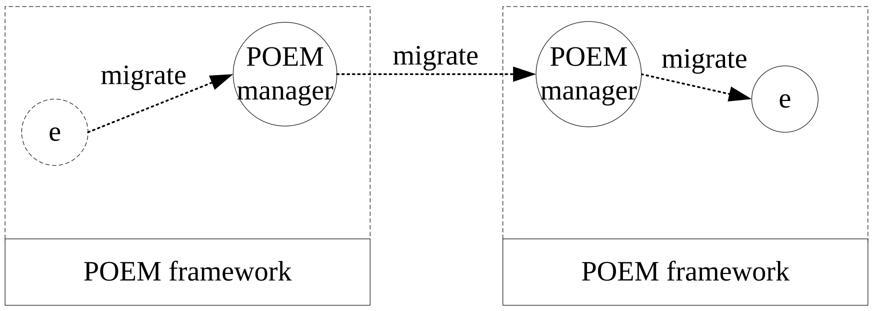
- The second application model is service offloading, which alleviates the cost in large data exchange. In the motivation example, Bob needs the video clips from Alice to run the image processing and filter out the vehicles. If service binding is applied, the images are sent from Alice to Bob and then Bob runs the image processing. The transferred image may be large or the transmission may be too frequent, which imposes cost on the data exchange. If image process is easy and the output is simply the vehicle profile in the image, then Bob may offload the image process PF to Alice to avoid the large amount of data transmission. In this way, the computation is migrated to the data location rather than service binding that feeds data to the computation. Another scenario where offloading may results in benefit is to migrate the resource intensive tasks from mobile devices to their ESSI. Both mobile devices and VMs in the clouds can be the surrogate containers to host the offloaded computation. Offloading is not always



(a) PF publishing and discovery.



(b) PF composition.



(c) PF offloading.

Figure 3.2: POEM Functionalities.

the best methods in any circumstances. Since the offloading itself has some cost and the devices acting as surrogates may run out of resources, offloading may lead to more cost or cannot run.

3.1.2 Execution Model

According to previous application models, there are three fundamental execution functionalities in the POEM system, as shown in Figure 3.2. There are two POEM frameworks running on two devices or machines. Each POEM framework has an identity so that they can form friendship relation, and the PFs on the framework can benefit from this friendship relation in the service discovery procedure. The items a to e represent PFs. Each subfigure describes one functionality:

- The Figure 3.2a describes how one PF discovers remote available PFs. PF *b* hosts a service and it publishes the service through the local POEM manager for remote PF to discover. Then PF *a* can discover the published service on remote side with local POEM manager's help. One prerequisite for *a* to discover and use service of *b* is that they are mutual friends, in other words they in each other's trusted list. PF *a* does not know that PF *b* is running on remote side because POEM manager pretends that *b* is running locally. Thus, a programmer does not need special treatments in coding when developing PF *a*.

- The Figure 3.2b presents how an application recruit a service provided by a remote PF. The PF *c* sends method invocation parameters, which are transferred by the POEM on local side and then on remote side, to the destination PF *d*. Then, the service result returns along the reverse route from *d* back to *c*. PF *c* also regards it is calling a local target *d* due to POEM transparent transfer, and *d* also thinks local *c* is calling it.

- The Figure 3.2c presents how one PF migrates to remote entity. A POEM PF initializes the migration process. There are two types of migrations: pull and push. In pull migration, the POEM PF on the right side sends request to left side POEM PF, and then the left side POEM manager fetches and transfers the target PF *e* to the right side. In push migration, the POEM PF on the left side transfers PF *e* to the remote side. The left side keeps the PF *e* active during transfer to provide the failsafe when the transferring is not successful.

The PF e code is copied to the right side POEM framework while keeping the left side PF e running to accept requests. While the right side PF e initializes completely and is ready for serving, the right side POEM framework advertises its e function with higher rank property, which will attract the new requests to the right side. The left side PF e will finish its current task and become idle. In case the right side POEM framework becomes offline, the left side idle PF can take over the new requests.

Service discovery in Figure 3.2a should be done before service binding in Figure 3.2b. When the PF needs some services that it does not have, it queries the OSGi framework to look up the desired service locally. If there is not, the POEM manager captures the query and search in its remote service registry. If there is matching remote service, the POEM manager triggers the service binding process. In this way, the combination of functionalities in Figure 3.2a and Figure 3.2b achieves the service composition application model.

Similarly, the service offloading application model can be achieved by combining Figure 3.2a, Figure 3.2c and Figure 3.2b. First, the service discovery is applied to see if the desired service is already available on the target framework or device. If it already exists, no offloading is necessary. If there is no such service on the destination POEM framework, the migration process can start. After the migration, the service runs on the target POEM framework. Once the service is migrated and active for accepting requests, the POEM manager on the target framework publishes the new service to its friend POEM framework instances. The computation is routed to the target POEM framework along with the service requests.

3.2 POEM Design

Figure 3.3 illustrates the overall design of the POEM system. The *POEM Manager* monitors local services, tracks service state change, maintains local PF repository and responds to remote service queries. Its *Networking* component maintains XMPP connections to XMPP peers that provides the communication and signaling infrastructure among mobile devices and their ESSIs. The *Identity and Trust* component identifies the POEM framework instance. It uses a JID as representative and registers the POEM framework instance to the XMPP server so that the POEM framework instance joins the XMPP network and can talk to other POEM framework instances. Besides, it also manages the friendship between POEM framework instances so that the trust relationship is constructed for secure surrogate. The *Publish and Discovery* component is responsible for publishing and broadcasting the local PF services. It is also responsible for replying the service queries and help the local PF to query the services on the remote POEM frameworks. The *Composition* component creates local proxy for remote service provider that responds to service request by transferring the request to the remote PF, and then getting the result to the local PFs. Based on a systematic decision model, POEM initiates the migration operations for PF offloading. The offloading action is accomplished by three components: *Context*, *Decision* and *Migration*. The *Context* component monitors the device resources and application running status to provide the environment and application profile and history to the Decision component to calculate the offloading decisions. The *Decision* component is a pluggable component, which is flexible to be replaced to upgrade the framework intellect. Although the POEM framework provides a default Decision component implementation, the proper offloading decision algorithms are suggested to be implemented for various scenarios due to the environment and task complexity.

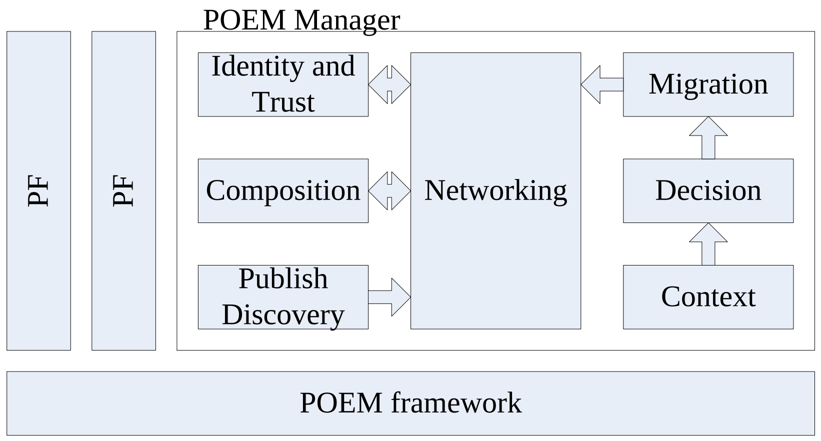


Figure 3.3: POEM Design and Components.

The *Migration* component fulfills the offloading decisions once the *Decision* component issues the offloading command. In the following sections, we describe details of each component within the POEM framework.

3.2.1 Distributed POEM Service Platform

POEM's networking and signaling system is deployed based on XMPP approaches. The communication between POEM entities (i.e., mobile devices and ESSIs) is full duplex compared to half duplex HTTP approach deployed by many web-based service frameworks. In a distributed execution environment, any entity can be both a client and a server at the same time, which is different from web-based service models where clients and servers are explicitly defined. Moreover, POEM inherits the XMPP trust and identity management framework, where every POEM entity is authenticated when joining the system and data transferred are also protected through cryptographic approaches. As a result, the PF offloading and PF compositions can utilize the XMPP trust management framework with fine-grained access control capabilities. Furthermore, POEM entities need to provide their presence information to indicate its availability information in real-time, which is as well used to indicate their service status. Finally, POEM must provide the support for secure file transfer, service discovery, and service composition, in which the XMPP provides the fundamental support to realize these features.

Identity and Trust Management

In the POEM framework, each POEM entity has a POEM manager that manages the local/remote PFs and it is uniquely identified by a unique JID when the user registered in the system. A user's JID could be shared among POEM entities, and it must be included in messages to identify the source/destination of the messages. The security features such as data privacy, integrity, user authentications, etc., can be easily incorporated by establishing a centralized trust authority, e.g., certificate authority or Kerberos-based key management, authentication, and authorization. We must note that using XMPP the social network-based trust can be easily established through friend lists (a.k.a., contact list). Using this feature, POEM entities can first request desired PF(s) to their friends before sending the discovery message to all other POEM users.

POEM Service Discovery and Publishing

POEM service discovery is designed based on XMPP service discovery protocol [Hildebrand *et al.*, 2008] and XMPP publish-subscribe extension [Millard *et al.*, 2010]. A PF may reside on a mobile device or its corresponding ESSI. The ESSI takes the responsibility to represent the mobile user for any PF related operations and the mobile device POEM Manager can frequently update its available PFs information to the ESSI. In this way, the main POEM service discovery, migration, and composition operations will not be flooded to end mobile devices. The ESSI POEM Manager also maintains the mobile device availability information and provides its reachability information to its trusted POEM peers. When the ESSI POEM Manager receives the service discovery message, it replies with its available PFs with the available remote service interfaces.

POEM Manager also monitors local service changes and notifies its friends. This is done through a publishing procedure. POEM Manager first registers a publish node (i.e., a virtual node in the XMPP server) under its JID. Thus, when local service status changes, POEM Manager can post the notice on its publish node and its friends get notified and update their PFs availability database.

POEM Service Composition

When POEM discovers service provided by remote POEM entities, it tries to create a proxy for that service so that remote PF can be used locally. POEM uses Java dynamic proxy technique to create proxy. Dynamic proxy requires that the target interface's Class instance must exist. To have remote service interface's Class instance in local OSGi framework instance, POEM fetches PF JAR file corresponding to the target service from remote POEM framework. POEM Manager installs the PF, and then the target Class instance is available and proxy generation is done.

POEM uses JavaScript Object Notation (JSON) over XMPP for service composition because JSON is lightweight and has abundant expression ability. The service proxy generated by POEM Manager captures local service requests that are then converted into JSON requests. Then the JSON request is sent to XMPP channel to the destination. The destination POEM Manager receives the JSON request and translates it to method invocation on service provider's object. It then returns the result in form of JSON back to the source POEM Manager. Then the JSON response is decoded and returned to calling object.

3.2.2 PF Offloading

When application decides to offload a service provider object and migrate it to cloud, POEM Manager chooses to send the object's byte code to cloud and start the object from byte code. How to choose POEM PFs to be migration is based on several conditions described as follows: First, thread migration solution is not adopted because some objects that exist in the same thread have to run on mobile device, such as user interfaces and sensors. Second, an application usually wants to migrate only the compute intensive operations rather than the whole thread. Third, object state is not maintained because the insight private details of the object to be migrated cannot be fetched due to Java security management. Our recent practice suggests that service implementation should be stateless, so that the object states will not bother POEM like <u>RE</u>presentational <u>S</u>tate <u>T</u>ransfer (REST) does [Fielding and Taylor, 2002].

Migration

The service provider object offloading process follows a three-step approach: First, the target PF JAR file is transferred to ESSI and started. Then, a proxy object is created to intercept and capture service request to remote target service. Finally, the PF containing target service provider object is stopped.

The migration happens according to the migration decision module command. POEM constructs the migration decision module as plug-in framework. The previous work [Wu *et al.*, 2013][Wu and Huang, 2014][Wu and Huang, 2015] on decisions can be applied. User can develop his own migration decision strategy plug-ins and install the strategy bundle into POEM, which not only provides the flexibility for user customized migration strategy but also scales the POEM intelligence.

PF Isolation

The migrated PFs are running in the surrogate POEM framework for providing service for its origination. These PFs may interact with the POEM framework and interrupt the PFs that belong to surrogate host. The PF isolation is required to protect the surrogate POEM framework and cease the potential attack from the migrated PF.

The POEM manager initializes a separate PF container for each friend who wants to offload his PF. The PF container is duplication of the surrogate host POEM framework. The only difference is that this nested PF container is empty and dedicate for the corresponding friend. The friend identity is stored and managed by identity manager. The surrogate host defines the accepted PF policies that are enforced by policy manager.

Connection Failsafe

The connection between mobile device and cloud is usually not stable as mobile device moves. When the connection is lost, POEM Manager restarts the PF that has been stopped in offloading process. The recovery process has the following two steps: First, the target PF is started. Then, the proxy service is unregistered and the proxy object is destroyed. The first step prepares for receiving service request. The second step destroys proxy, which makes the target service provider object be the first in the ranking order to receive service request.

Offloading Decision

POEM has two types of offloading decisions. The default type is "the POEM decides when to migrate component", and the other is "the user decides right migration moment". To enable PF migrations, POEM publishes local migration service API. Once this API is called, POEM executes migration process. POEM design leverages the plug-in concept. The offloading decision module can be plugged in to enable the different offloading strategies to be applied onto POEM framework. More offloading strategies are in Chapter 4.

3.3 POEM Implementation

As Figure 3.2 shows, the key component is the POEM manager bundle which accomplishes the service discovery and publish, service composition and service offloading features. This section describes the implementation details of the POEM Manager OSGi bundle as well as other implementation issues including the seamless offloading procedure.

3.3.1 POEM Manager OSGi Bundle Implementation

POEM Manager consists of several objects as shown in Figure 3.4. They are categorized as three sets - XMPP connection and related listeners, PF context and related listeners, and proxy and migration management. The three object sets represent three POEM functional sets: XMPP connection set represents remote POEM framework; PF context set represents local POEM framework; and proxy and migration management represent core POEM logic and operation that connect the other two parts.

XMPP connection object maintains three XMPP managers that manage service discovery, publish-subscribe, and file transfer separately. Besides, it also maintains a *roster* that publishes local presence and a *publish node* that local service change notification is posted on. There is a set of listeners registered with XMPP connection. They are noticed when corresponding events occur. *Roster listener* tracks friends' presence and update proxy pool accordingly. *Item event listeners*, one listener for one friend, wait for friends' service change notice and update proxy pool accordingly.

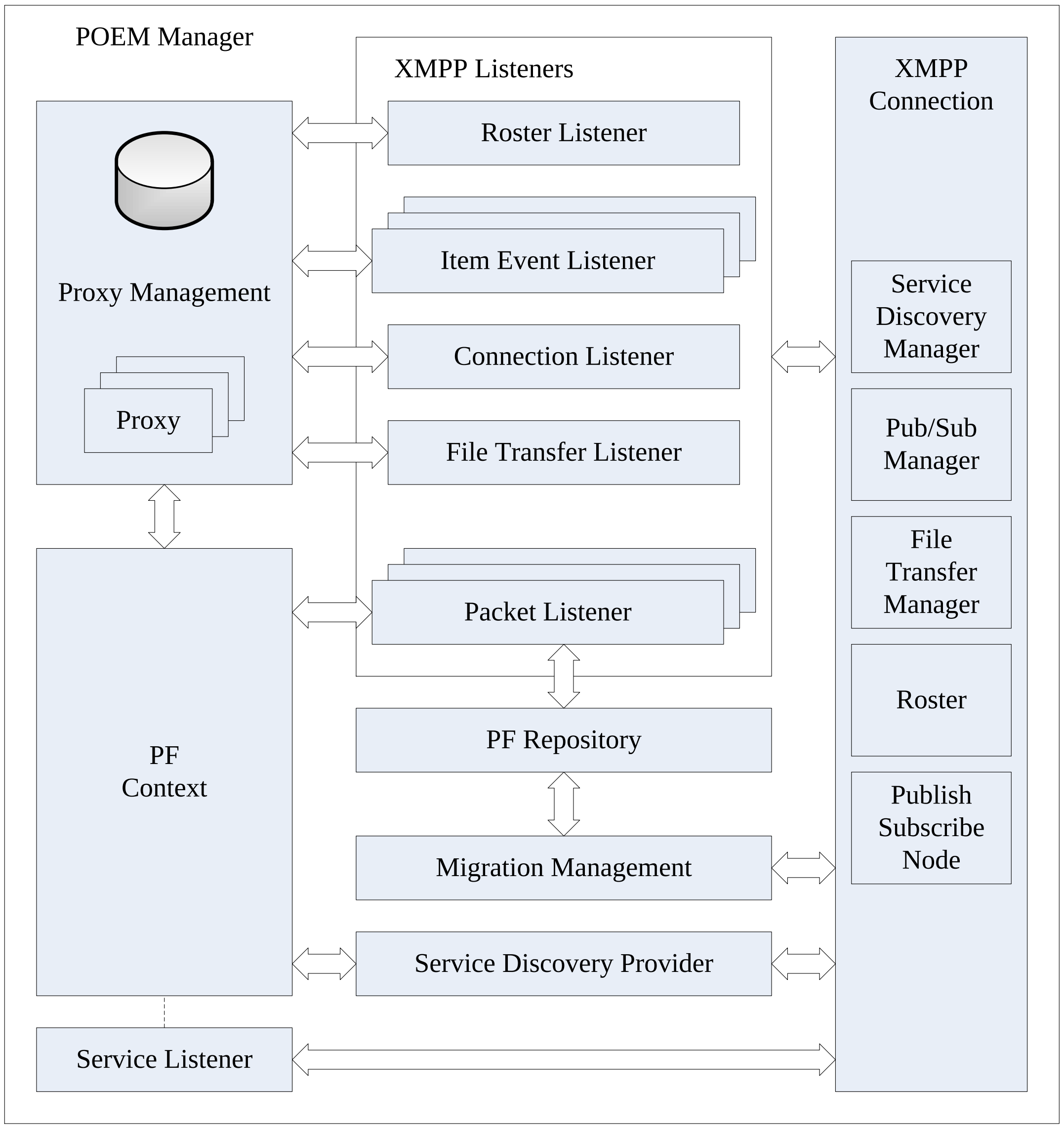


Figure 3.4: POEM Manager Details.

Connection listener monitors connection status and executes robustness strategy. *File transfer listener* handles file transferring. *Packet listeners* handle *iq* packets defined in POEM name space between POEM PFs. *Service discovery provider* responses to remote service discovery by querying PF context.

PF context object set includes *PF context* and *Service listener*. *PF context* handles interaction to POEM framework or OSGi framework. Since POEM manager sits on top of OSGi container, it has to interact with OSGi framework. The PF context is the bridge that connects the POEM manager and the OSGi container. *Service listener* monitors local service change that is then published to the remote nodes maintained by XMPP connection.

The proxy and migration management objects include *Proxy management*, *Migration management* and *PF repository*. *Proxy management* contains a database and a *proxy* pool. It memorizes remote service status and local proxy status in database, and provides proxy generation and recycling methods. *Migration management* implements migration service registered by POEM Manager. *PF repository* provides JAR file source for file transfer request. The OSGi framework already has a bundle repository for all the installed bundles. POEM reuses the OSGi framework bundle repository and only caches the remote bundles. When exporting the local service to remote POEM instance, the *Migration management* fetches the bundle JAR file from the OSGi framework bundle repository. When importing the service from remote POEM instance, the *Migration management* gets the bundle JAR file from *XMPP connection* object and installs the bundle. The POEM framework does not have to maintain the bundle repository itself.

Service Publish and Discovery

Roster listener and *item event listener* monitors remote service notice and update local proxies. *Roster listener* is noticed when remote POEM Manager goes online or offline. If remote POEM Manager goes online, it tries to discover all the available remote services. If remote POEM Manager goes offline, it tries to recycle local proxies. *Item event listener*, instead of being noticed by presence and trying to actively discover service, is noticed when remote POEM Manager posts service status change. *Service listener* is responsible for publishing the service change.

Service Composition

After *roster listener* or *item event listener* discovers the remote service, it fetches the corresponding JAR files from remote framework. After it gets JAR file, it installs them but does not start them. Then *proxy management* create proxies according to discovered service name. The *proxy* is constructed by Java dynamic proxy technique [Eckel, 2006] that requires class information provided by installed JAR file and service name provided by service discovery or publish-subscribe notice.

When local PF consumes remote service, *proxy* intercepts the request. Then *proxy* wraps the request into JSON object and inserts the JSON object into XMPP *iq* packet that is sent to remote POEM Manager. When remote *packet listener* receives the *iq* request packet, it parses the JSON object and calls the local service and reply to requestor POEM Manager in the same way. The requestor *proxy* collects the *iq* response, parse it and return to local requestor.

POEM on Android

For mobile device side, POEM is implemented on Android devices. Android uses a different byte code format from normal Java byte code. Normal Java JAR file can be recognized by Android after DEX and AAPT operations [Chen *et al.*, 2011]. Android SDK provides the necessary tools. DEX tool collects class information and generate classes.dex file. AAPT tool injects classes.dex file into the original JAR file. After these two steps, the JAR file can be recognized by Android.

3.3.2 Seamless Offloading

POEM Manager registers a service with an Java interface that contains a method to do service migration. Service migration involves two framework instances that are source framework and destination framework. The offloading process can be illustrated using the following application scenario. The source is device 1 and the destination is an ESSI. The migration method is called on device 1. Service name and destination XMPP identity are passed to the migration method. The migration process consists of five steps as follow. First, a migration notice is sent by device 1 to the ESSI. Along with the migration notice, the PF JAR file that owns the indicated service is transferred from device 1 to the ESSI. Second, POEM Manager in the ESSI starts the PF. When PF is running, services including the indicated service are registered. Third, POEM Manager in the ESSI is notified with service changes in last step. it unregister existing proxy under the same service name. Then it publishes the new services to the ESSI's publish node. At this point, both sides have the running PF that provides services to local PFs. Fourth, POEM Manager on device 1 is notified due to the publishing in last step. it creates the proxy for the published services with a higher ranking. Then it stops the local PF. At this point, the PFs on device 1 are consuming services provided by the ESSI. The sequence diagram of migration process is shown in Figure 3.5.

Besides device 1 and the ESSI, a third framework instance on device 2 is using the service being migrated. When POEM Manager in the ESSI signals the new service,

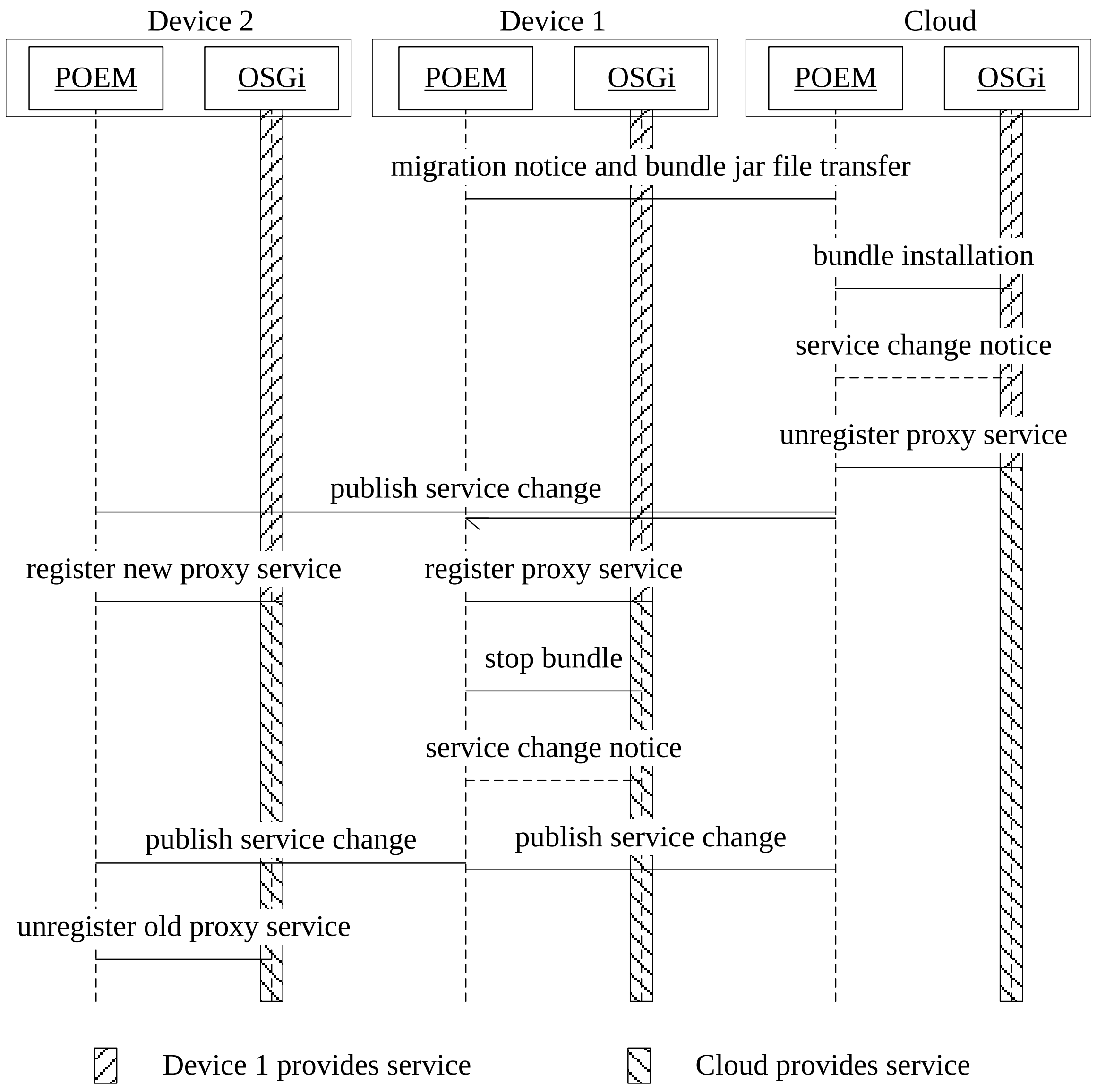


Figure 3.5: PF Migration Sequence.

POEM Manager on device 2 creates proxy for the new service with a higher ranking as device 1 does. When POEM Manager in the ESSI signals the service recycling, POEM Manager on device 2 recycles the proxy for that service. Other PFs on device 2 are not disturbed during the process.

3.4 Evaluation

This section describes POEM performance evaluation through both macro-benchmarks and micro-benchmarks. Then migration evaluation is then followed.

3.4.1 Methodology

The POEM Manager is implemented on Felix [Gédéon, 2010] OSGi implementation version 4.0.3. Mobile application that contains a Felix OSGi framework instance that hosts POEM Manager runs on Android Motorola phone A855. The phone's parameters are 600MHz CPU and 256M memory. The Android version is 2.2.3. The virtual machine is with 1GHZ CPU and 512M memory, which runs Ubuntu 11.10.

Four applications are used to evaluate the POEM performance. They are Fibonacci sequence generator, N-Queens puzzle, nested loop and permutation generator. The Fibonacci application generates Fibonacci sequence in a recursive manner. Its time complexity is $O(2^n)$ and its stack usage is high due to recursive algorithm. The N-Queens application calculates all solutions for input chessboard size. Its time complexity is $O(n^2)$ and its stack usage is also high due to recursive algorithm. The nested loop application contains a six layer loop which leads to time complexity $O(n^6)$. The permutation application's time complexity is $O(n!)$ and uses little memory. Experiment result is obtained by running the application 50 times for every scenario and averaged. Between two consecutive executions there is a pause of 1 second.

The experiments are run under two scenarios:

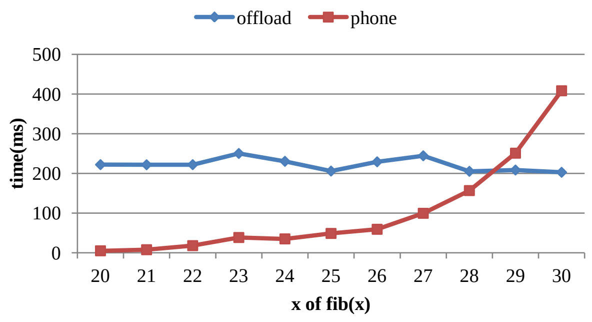
- Phone: Applications are run only in phone.
- WiFi: Phone is connected to the ESSI through WiFi.

The WiFi connection has averaged latency of 70 ms, download bandwidth of 7 Mbps, and upload bandwidth of 0.9 Mbps. Ping is used to report the average latency from the phone to the ESSI, and Xtremelabs Speedtest, downloaded from Android market, is used to measure download and upload bandwidth.

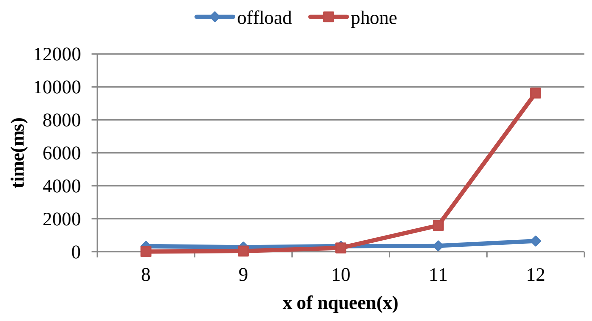
3.4.2 Macro-benchmarks

For typical input parameter values, four applications are run on phone and in the ESSI separately. The application running time is recorded in Table 3.1. By subtracting time on phone and in the ESSI, the max speed up is put in the last column of the table. However, the max speed up is seldom achieved due to cost of communication and proxy. This cost changes little while offloading benefit changes much, so there should be some point when the benefit of offloading surpasses its cost giving application net gain.

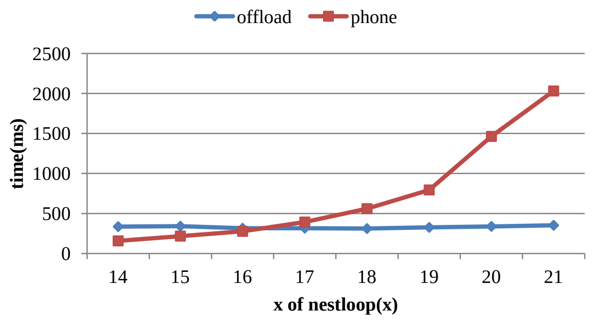
Fibonacci application takes a sequence index number and calculates the corresponding number in the Fibonacci sequence. Figure 3.6a shows execution time of Fibonacci application. The intersection of execution time on phone and WiFi offloading is the Boundary input value (BIV) [Kosta *et al.*, 2012] that shows the offloading benefit starting point. N-Queens application takes chess board size and calculates all solutions and return solution number. Figure 3.6b shows execution time of N-Queens application. The execution time on phone rises dramatically as the chessboard size increases one scale. Offloading offers benefit after chess size is larger than 10. Nested loop application takes loop times and execute loop without memory operation. The



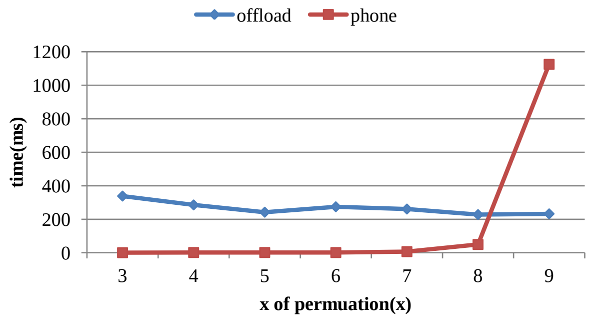
(a) Execution time of Fibonacci application.



(b) Execution time of N-Queens application.



(c) Execution time of nested loop application.

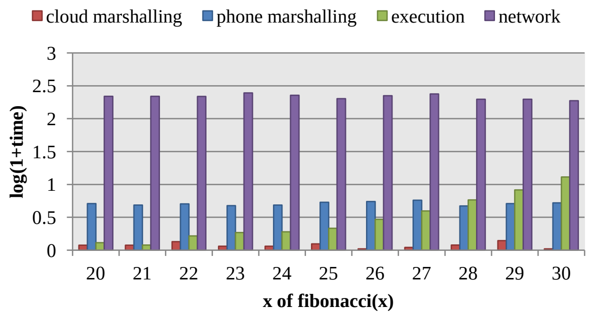


(d) Execution time of Permutation application.

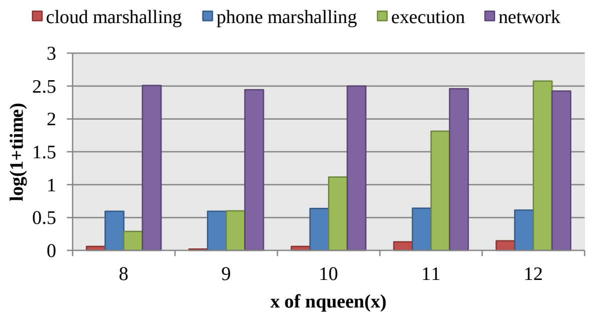
Figure 3.6: Execution Time of Four Test Applications.

Table 3.1: Max Speed Up with Offloading

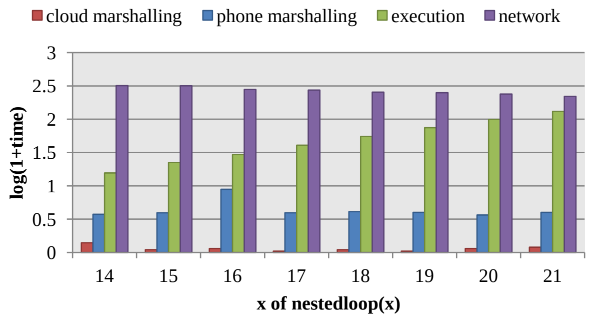
Case	Input	Phone (ms)	Cloud (ms)	Max speed up (ms)
Fibonacci	26	59.25	2	57.25
	27	99.5	3.05	96.45
	28	156.75	5	151.75
	29	251	7.65	243.35
	30	408.25	12	396.25
N-Queens	8	11	1.1	9.9
	9	39.75	3.05	36.7
	10	222.75	12.2	210.55
	11	1593.5	64.4	1529.1
	12	9630.25	377.2	9253.05
Nested loop	14	157	15.05	141.95
	15	332	21.55	310.45
	16	276.75	28.6	248.15
	17	392.5	39.85	352.65
	18	560.25	54.35	505.9
Permutation	5	1.25	0.25	1
	6	1	0.25	0.75
	7	6.5	0.4	6.1
	8	49.25	2.05	47.2
	9	1124.75	12.1	1114.65



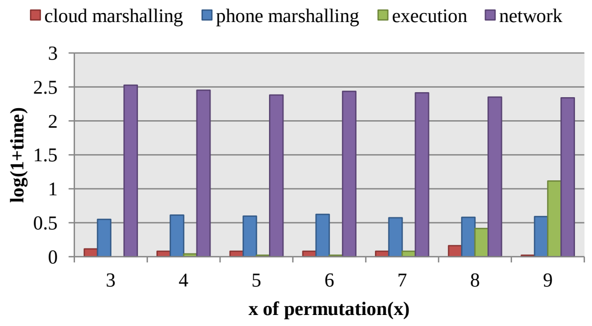
(a) invocation time of Fibonacci application.



(b) invocation time of N-Queens application.



(c) invocation time of nested loop application.



(d) invocation time of Permutation application.

Figure 3.7: Service Invocation Time of Four Test Applications.

execution time on phone is convex, which means it is less than exponential increase compared to the above two applications that requires both computing and storage. The execution time of offloading increases slowly. The Permutation application takes a max number N and returns count of prime number within the range (1,N). The prime number searching algorithm used is Permutation algorithm. The execution time increases on phone, however the execution time for offloading approach almost remains same.

The offloading line of four applications is increasing slowly compared to phone line. As the phone line starts from a low point, which indicates the application runs fast when input is small, the offloading line and phone line intersects finally. Comparing offloading line and the ESSI execution time column in Table 3.1, the slow increase is reasonable due to execution time increase slowing in the ESSI as well. Besides, the starting point of offloading line is higher than phone line, so there must be cost for remote method invocation.

3.4.3 Micro-benchmarks

This experiment measures service invocation time. This time is measured on phone where is service consumer side. The remote service consuming time consists of three parts: marshaling time of both consumer and provider sides, network transfer time and actual execution time. The result is shown in Figure 3.7.

Figure 3.7 shows time against different input parameters. From the table, the actual execution time is similar to the execution in the ESSI of column the ESSI in table 3.1. At the beginning, execution time is nearly zero. The execution time increases along with input parameter value increases. Figure 3.7 shows that marshaling time is relatively small compared to network delay. Figure 3.7 also shows that the main cost for remote method invocation is network delay around BIV point.

Table 3.2: Service Migration Time

Cases	migration time (ms)
Fibonacci	272
N-Queens	335
Nested loop	290
Permutation	304

And marshaling time and network time against different input parameters are approximately identical. The marshaling and network cost decides the start points of offloading line in Figures 3.6a-3.6d. And execution time decides the trend of those offloading line. If the network delay or the marshaling is reduced in some situation, the offloading line will drop and then BIV point will go to left, which means the range of benefit increase and application components are supposed to be offloaded to the ESSI. In another perspective, if component's ratio of computation cost to network cost increases, it is better to offload that component to the ESSI.

Besides service invocation time, the proxy generation time is also measured. The proxy generation time indicates POEM initialization time, which is paid once at starting POEM Manager.

3.4.4 PF Migration

This experiment measures PF migration time. PF migration time period starts when service migration command is issued and ends when proxy for migrated service is available. The result is in table 3.2 which shows that the migration time is nearly same for the tested four applications. This is reasonable because the migration time is mainly the time of transferring PF bundles on the network and these four PF bundle sizes are similar.

Chapter 4

MOBILE CLOUD OFFLOADING STRATEGIES

Based on the POEM framework in the Chapter 3, various offloading strategies have been developed to make the intelligent offloading decisions in the various context and environment according to Section 3.2.2. This chapter discusses various offloading strategies of different offloading modes and in different environment assumptions.

Offloading can happen between one mobile device and one VM, or between several mobile devices and several VMs. In addition to the offloading topology, offloading can happen once or multiple times in a series. The most simple offloading mode is one time one-to-one offloading. This type usually happens between the mobile device and its corresponding VM. One-to-one offloading is simple due to its explicit surrogate target VM, while many-to-many offloading may lead to better performance due to its best choice of surrogate target. For long time application running, the environment or the context may change during the application life time, thus the offloading may happen multiple times to adjust the offloading topology to give best outcome all the time.

The offloading is not a zero-cost business. To achieve a complete offloading process, the mobile application first has to contact the surrogate target to initiate the offloading process. The offloading process includes the code migration, the service migration and the service composition procedures. These procedures consume network bandwidth, CPU cycles, storage and mobile device energy as well. Due the offloading cost, the offloading does not always provide benefit for the mobile applications.

The common objectives of offloading are reduce computation time and save energy on the mobile devices. For the computation intensive application, offloading may be a good candidate solution, because the network and energy are saved and the computation is offloaded to the powerful surrogate. In the other hand, offloading may not be a good idea for the communication intensive applications.

The following sections will discuss the benefit and cost for several offloading topology in different circumstances. Section 4.1 starts with the simple one time one-to-one offloading and then applies the network varation to the model. Section 4.2 presents the many-to-many topology and multiple offloading objectives. Section 4.3 presents multiple offloading series in time horizon.

4.1 Simple Offloading Model and Network Unavailability Impact

The offloading relies on the network connecting the mobile device to the VM. If the network is not stable, which is very usual in the mobile scenario, the offloading decision may have very different results, thus the network condition has to be taken into consideration. To address the problem of offloading decision making in network with some unavailability, an offloading decision module that produces offloading decision that is resistant to network unavailability is presented in this section. The unstable network is modeled as an alternating renewal process that provides statistic information of connection duration. The proposed offloading decision module monitors network connection states and durations that are recorded in a history buffer. Then it calculates application partition that is aimed to give benefit in network with low availability. And it finally validates the partition and outputs the offloading decision. The offloading decision module contains two key components. The first is for partitioning application components into smart phone and cloud sides based on unstable network assumption. The second is for validating the application partition by comparing the possible unavailability of the partition and the observed network unavailability. The proposed offloading decision module can be put into mobile cloud application to provide application offloading decisions with the consideration of network unavailability.

In summary, the contributions of presented research are highlighted as follow:

- *mobile cloud collaboration model considering network unavailability:* The proposed scheme models the application execution time considering unstable network situation. The unstable network is modeled as an alternating renewal process. Then application execution time and energy consumption are analyzed in both ideal network and unstable network.

- *application partition algorithm:* The proposed algorithm utilizes application information to find the application partition that can achieve both less execution time and less energy consumption when network availability is low. The algorithm works in a heuristic manner.

- *bayesian decision approach:* The partition given by above partition algorithm is validated by comparing tolerated unavailability of given partition and observed network unavailability. Since the network states estimated based on history observation are not fully trusted. The final offloading decision is made by a bayesian decision approach to mitigate the possible error.

- *offloading decision module:* The proposed solution provides a module that can be plugged into mobile cloud applications to enable decision making in unstable network scenario. The offloading decision module consists of four parts: observation buffer, application partition component, network state predictor and partition validation component.

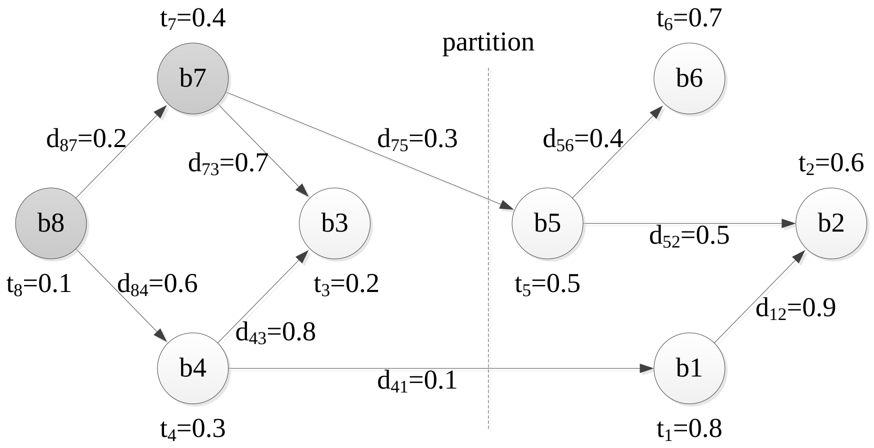


Figure 4.1: Application Graph Example.

4.1.1 System and Models

Mobile cloud consists of mobile devices and cloud. There is usually one to one mapping between mobile devices and VMs in cloud [Huang, 2011]. The mobile cloud application is constructed as a set of components or bundles. Some components can run on either mobile device or VM, while the other components, such as user interface and sensors, have to run on mobile devices. The major offloading objectives are saving application execution time and energy consumption on mobile device.

Application Model and Ideal Network Model

The application is presented as a directed acyclic graph $G = \{B, E\}$ where every vertex is a bundle and every edge is data exchange between bundles [Giurgiu *et al.*, 2009]. Each bundle has an attribute indicating whether it can be offloaded. The unmovable bundles are marked as *local* which means these bundles cannot be offloaded due to application requirements. Let m be the total count of bundles in the application, then the initial bundle set is $B = \{b_i \mid i \in [1, 2, \dots, m]\}$ and the edge set is $E = \{e_{ij} \mid i, j \in [1, 2, \dots, m]\}$ where e_{ij} represents directed data transfer from b_i to b_j. Let n be the count of movable bundles and $n \leq m$. A configuration c [Giurgiu *et al.*, 2009] is defined as a tuple of partitions from the initial bundle set, $< B_{phone}, B_{cloud} >$, where $B_{phone} = \{b_i \mid i \in [1, 2, \ldots, k]\}$ has k bundles and $B_{cloud} = \{b_i \mid i \in [1, 2, \ldots, s]\}$ has s bundles. And they satisfy $B_{phone} \cap B_{cloud} = \emptyset$ and $B_{phone} \cup B_{cloud} = B$. The bundles that are marked as *local* are initially put in the set B_{phone} and cannot be moved. An example is shown in Figure 4.1 where unmovable bundles are marked as grey and dot line indicates configuration. The bundles on the left side are B_{phone} and the bundles on the right side are B_{cloud}.

Execution Time For a given task, bundle b_i has an attribute t_i indicating its computation time on smart phone. And edge e_{ij} is associated with an attribute d_{ij} indicating the transferred date size from bundle i to bundle j. These value can be measured or estimated from collected application and device data. Total time of running task only on smart phone is

$$t_{phone} = \sum_{b_i \in B} t_i \tag{4.1}$$

where data exchanges between bundles are not counted as they happen locally and cost little time compared to time of data exchange over network. For a particular configuration c, offloading rate p [Ou *et al.*, 2008] is defined as the proportion of offloaded task to all task in terms of computation time. Then, task proportion is the same as time proportion due to the same processing capability on the same mobile device $p(c) = (\sum\limits_{b_i \in B_{cloud}} t_i)/t_{phone}$, and $p(c)$ satisfies $0 \leq p(c) \leq 1$. Then, the computation time on smart phone is

$$\begin{aligned} t^{phone}_{computation}(c) &= \sum_{b_i \in B_{phone}} t_i = (\sum_{b_i \in B} - \sum_{b_i \in B_{cloud}})t_i \\ &= (1 - p(c))t_{phone} \end{aligned} \tag{4.2}$$

Assume the cloud is q times faster than smart phone, thus the time consumption in cloud is q times less than the time spent on mobile device [Xian *et al.*, 2007], which is $t_i^{cloud} = t_i/q$. The computation time in cloud is

$$\begin{aligned} t^{cloud}_{computation}(c) &= \sum_{b_i \in B_{cloud}} t_i^{cloud} = \frac{1}{q} \sum_{b_i \in B_{cloud}} t_i \\ &= \frac{p(c)}{q} t_{phone} \end{aligned} \tag{4.3}$$

Thus the total computation time is:

$$\begin{aligned} t_{computation}(c) &= t^{phone}_{computation}(c) + t^{cloud}_{computation}(c) \\ &= (1 - (1 - \frac{1}{q})p(c))t_{phone} \end{aligned} \tag{4.4}$$

A typical offloading process works as follows. Initially, the application starts on smart phone and all components run locally. Then, the application may offload some components to remote virtual machine. These offloaded bundles run in cloud remotely. However they need to communicate with the bundles resident on smart phone. Thus they have to exchange data through network. Assume network bandwidth is w, then the network delay is the sum of delays in both data transfer directions:

$$t_{network}(c) = \sum_{\substack{b_i \in B_{phone} \\ b_j \in B_{cloud}}} \frac{d_{ij}}{w} + \sum_{\substack{b_i \in B_{cloud} \\ b_j \in B_{phone}}} \frac{d_{ij}}{w} \tag{4.5}$$

In an ideal network environment, the total execution time for a given configuration c is the sum of computation time and network delay.

$$t(c) = t_{computation}(c) + t_{network}(c) \tag{4.6}$$

The offloading benefit of execution time comes from the trade of $t_{computation}$ and $t_{network}$. The computation part saves execution time because the cloud processing capability is powerful than mobile device. However, the offloading has to pay network delay, which counteracts the computation time saving. For computation intensive applications whose computation time saving is much larger than network delay, the offloading benefit is obviously seen.

Energy Consumption Two hardware modules of mobile device are involved in the energy consumption estimation: CPU and Radio Frequency (RF) module. The other modules, like display, audio, GPS etc., are not considered because the components that interact with theses modules have to run on mobile device locally. Both energy consumption on CPU and RF module can be further separated into dynamic part and static part [Wang *et al.*, 2013]. When hardware module is in idle state, the energy consumption is corresponding to static part. When hardware module is in active state, more energy is consumed, which is corresponding to dynamic part. Assume the power of CPU in idle state is K_{CPU}^{sta} and the power of CPU in active state is $K_{CPU}^{sta} + K_{CPU}^{dyn}$. The energy consumption of CPU is:

$$P_{CPU}(c) = K_{CPU}^{sta} t(c) + K_{CPU}^{dyn} t_{computation}^{phone}(c) \tag{4.7}$$

Similarly, let K_{RF}^{sta} and K_{RF}^{dyn} be the power of RF module in idle and active state separately. The energy consumption on radio frequency module is:

$$P_{RF}(c) = K_{RF}^{sta} t(c) + K_{RF}^{dyn} t_{network}(c) \tag{4.8}$$

Thus, the total energy consumption is:

$$P(c) = P_{CPU}(c) + P_{RF}(c) \tag{4.9}$$

If offloading is not applied, only CPU consumes energy and its active period is the whole execution time. The total energy consumption of running tasks only on smart phone is:

$$P_{phone} = (K_{CPU}^{sta} + K_{CPU}^{dyn}) t_{phone} \tag{4.10}$$

The offloading influences energy consumption of mobile device in two aspects. First, the mobile device may save energy because mobile device does not pay for the energy consumption corresponding to the tasks that are offloaded and completed in cloud.

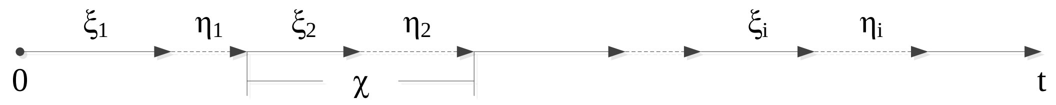


Figure 4.2: Network Unavailability Model.

Second, the data exchange between application components are now fulfilled by networking instead of local procedure invocation, which leads to energy cost for sending and receiving packets. Similarly to time benefit of offloading, the computation intensive application may obtain obvious energy benefit when computation tasks are offloaded to save large CPU energy consumption and network energy consumption is small.

Model and Impact of Network Unavailability

The connection between mobile device and cloud is usually not stable due to mobility of devices. When mobile device moves out of wireless coverage, it loses connection to cloud. The mobile device continues to make attempts to reconnect to cloud when the network is unavailable. When it gets into coverage again, the connection resumes. As mobile device moves, the connection state changes as *on*, *off*, *on*, *off* $\ldots$, which can be modeled as an alternating renewal process.

Figure 4.2 shows how network availability changes along with time coordinate. Solid line represents network is available, while dash line represents network is unavailable. Two network states alternate with each other. One *on* duration and one *off* duration form a cycle. The *on* state duration is denoted as ξ and the *off* state duration is denoted as η. $\{\xi_i, i = 1, 2, \ldots\}$ is independent and identically distributed (i.i.d.), and so is $\{\eta_i, i = 1, 2, \ldots\}$. And ξ_i and η_j are independent for any $i \neq j$, but ξ_i and η_i can be dependent [Pham-Gia and Turkkan, 1999]. The cycle duration is denoted as χ and $\chi_i = \xi_i + \eta_i$ where $i = 1, 2, \ldots$. The proportion of *on* duration in any individual cycle is a random variable denoted as $\rho = \xi/\chi$.

Execution Time When the network is unavailable, the application has to wait because phone cannot send input to cloud and cannot retrieve output from cloud either. The application resume the execution after the network becomes available again. The total execution time is prolonged according to the proportion of ρ:

$$t'(c) = \frac{t(c)}{\rho} \tag{4.11}$$

The offloading gives time benefit when $t'(c) < t_{phone}(c)$. In ideal network environment, $\rho = 1$ and $t'(c) = t(c)$. $t'(c)$ raises to infinity when ρ decreases from 1 to 0. At some point, the benefit disappears finally. We define this point as critical value of ρ for time benefit:

$$\rho'_{time}(c) = \frac{t(c)}{t_{phone}} \tag{4.12}$$

And the time benefit is:

$$\Delta t(c) = t_{phone}(c) - t'(c) \tag{4.13}$$

when $\rho > \rho'_{time}(c)$.

Energy Consumption During time period $t'(c)$, the computation time $t'_{computation}(c)$ and network time $t'_{network}(c)$ are the same with $t_{computation}(c)$ and $t_{network}(c)$ in ideal network environment because computation and data transfer only work properly when network is available as ideal network. The CPU active time period is the same as that in ideal network model because the given task is the same. However, the CPU idle time period is the whole execution time that is different from that in ideal network model. Thus, the energy consumption for CPU is:

$$P'_{CPU}(c) = K^{sta}_{CPU} t'(c) + K^{dyn}_{CPU} t^{phone}_{computation}(c) \tag{4.14}$$

The RF module is active even when the network is unavailable because it continues scanning the available network to resume the connection. Thus, the active time period for RF module is $t'(c) - t_{computation}(c)$. The energy consumption for RF is:

$$P'_{RF}(c) = K_{RF}^{sta} t'(c) + K_{RF}^{dyn}(t'(c) - t_{computation}(c)) \tag{4.15}$$

Thus, the total energy consumption is:

$$P'(c) = P'_{CPU}(c) + P'_{RF}(c) \tag{4.16}$$

The offloading gives energy benefit if $P'(c) < P_{phone}$. As ρ decreases, both $P'_{CPU}(c)$ and $P'_{RF}(c)$ increase. Similarly, the critical value of ρ for energy is defined as the point where energy benefit disappears:

$$\begin{aligned}\rho'_{energy}(c) = (K_{CPU}^{sta} + K_{RF}^{sta} + K_{RF}^{dyn})t(c)/(P_{phone} \\ -K_{CPU}^{dyn} t_{computation}^{phone}(c) + K_{RF}^{dyn} t_{computation}(c))\end{aligned} \tag{4.17}$$

And the offloading energy benefit is:

$$\Delta P(c) = P_{phone}(c) - P'(c) \tag{4.18}$$

when $\rho > \rho'_{energy}(c)$.

Problem Formulation When network availability ρ is greater than the larger one of $\rho'_{time}(c)$ and $\rho'_{energy}(c)$, both time and energy benefit are obtained. We define the critical value of ρ is:

$$\rho'(c) = \max\{\rho'_{time}(c), \rho'_{energy}(c)\} \tag{4.19}$$

The offloading problem with network unavailability consideration is to find the application partition c to minimize $\rho'(c)$:

$$\min \ \rho'(c) \tag{4.20}$$

while both time and energy benefit exist:

$$\rho > \rho'(c) \tag{4.21}$$

where ρ is the current network availability estimated based on observations. The c satisfying (4.21) may not exists when ρ is too low. In this situation, the application should not offload any components to cloud. The solution given by (4.20) is the best partition that tolerates the network unavailability, and it may give benefit when current network availability ρ goes worse.

4.1.2 Algorithm and Solution

The bundle computation time t_i's form a vector $\mathbf{t}$ of m dimensions. The data size d_{ij}'s form a square matrix $\mathbf{D}_{m\times m}$. If there is no edge from b_i to b_j, then d_{ij} is set to 0.

The configuration c can be represented as a vector $\mathbf{x}$ of m dimensions where x_i indicates whether b_i should be offloaded. $x_i = 1$ means b_i should be offloaded to remote cloud, and x_i=0 means b_i should be kept on smart phone locally. For b_i that cannot be offloaded, x_i is set to 0 initially and does not change. Vector $\mathbf{x}$ has m elements in which n elements are variables and the others are 0s. For simplicity, all 0s are put at the end of $\mathbf{x}$.

Let $\mathbf{1}$ be a column vector whose elements are all 1s, then $t_{phone} = \mathbf{t}^T\mathbf{1}$. Offloading rate $p(c)$ is now $p(\mathbf{x}) = \mathbf{t}^T\mathbf{x}/\mathbf{t}^T\mathbf{1}$. And $t_{network}(c)$ is $t_{network}(\mathbf{x}) = ((\mathbf{1}-\mathbf{x})^T\mathbf{D}\mathbf{x} + \mathbf{x}^T\mathbf{D}(\mathbf{1}-\mathbf{x}))/w$. Thus $t(c)$ is finally function of $\mathbf{x}$, which is $t(\mathbf{x})$.

The objective of offloading decision is to find configuration $\mathbf{x}$ satisfying (4.20). This is a 0-1 Integer Programming (IP) problem.

Application Partition

The solution space for configuration $\mathbf{x}$ is 2^n, which means it costs a lot of time to search the optimal solution. To find proper $\mathbf{x}$ within acceptable time, we propose an ABC (Artificial Bee Colony) [Karaboga and Akay, 2009] based algorithm. The colony consists of three types of bees: employed bees, onlooker bees and scout bees. The bee that goes to food source visited by it before is named employed bee. The bee that waits on the dance area for making a selection of food source is called onlooker bee. And the bee that carries random search for discovering new food source is named scout bee. The food source is a possible solution $\mathbf{x}$, and every bee can memorize one food source. It is assumed that there is only one employed bee for each food source. The memory of employed bees is considered as the consensus of the whole colony, and the food sources found by onlooker bees or scout bees merge into employed bees' memory in algorithm. Assume the number of employed bees is N and the number of onlooker bees is M ($M > N$). And let MCN be Maximum Cycle Number. The algorithm overview is shown in Algorithm 1.

Algorithm 1 Application partition algorithm overview.

```
1: Initialize employed bees
2: $cycle \leftarrow 1$
3: repeat
4:     Produce new solution for employed bees
5:     Apply greedy selection process for employed bees
6:     Determine probabilities, and assign $M$ onlooker bees to $N$ employed bees
       accordingly
7:     Produce new solution for onlooker bees
8:     Apply greedy selection process for onlooker bees
9:     Determine abandon solution, if exists, and replace it with scout bee
10:    Memorize the best solution so far
11:    $cycle \leftarrow cycle + 1$
12: until $cycle = MCN$
```

At the first step, the algorithm generates a random initial population X ($cycle = 0$) of N solutions where the population size is the same as number of employed bees. Based on this initial generation, the algorithm starts to evolve the generation in cycles. The evolution repeats until the cycle number reaches limit MCN. The algorithm outputs the best solution, denoted as $\mathbf{x}_{best}$, ever found in all cycles.

In the cycle, three types of bees work in sequence. The details of three type bees' actions are shown in Algorithm 2. Employed bees produce new solutions by two local search methods:

Flip employed bee randomly flips one element in the vector $\mathbf{x}$.

Swap employed bee randomly flips two elements of different values in the vector $\mathbf{x}$, which is equivalent to swapping two different elements in that vector.

Each employed bee evaluates the fitness of its original solution $\mathbf{x}$, new found $\mathbf{x}_{flip}$ and $\mathbf{x}_{swap}$ by (4.19). Then, each employed bee memorizes the best one of these three food sources and forgets the others.

Onlooker bees watch employed bee dancing, and plan the preferred food source. Onlooker bees record critical values of all food sources and calculate the probability for i^{th} food source as below:

$$p_i = \frac{1/\rho'(\mathbf{x}_i)}{\sum_{j=1}^{N}(1/\rho'(\mathbf{x}_j))} \tag{4.22}$$

Intentionally, the lower food source's critical value is, the more likely onlooker bee would like to go. The onlooker bees chooses the food source $\mathbf{y}$ randomly according to its probability. Since $M > N$, several onlooker bees may follow the same employed bee and choose the same food source. Then each onlooker bee applies the same local search methods used by employed bees previously to explore new neighbour solutions, and picks the best one of the three. After all onlooker bees update their solution, each employed bee compares its solution with its followers' solutions, and picks the best one as its new solution.

In our algorithm, only one scout bee is used. This scout bee randomly generates vector $\mathbf{z}$ and compares $\mathbf{z}$ to the worst solution of employed bees. If this random generated $\mathbf{z}$ is better than the worst solution of employed bees, the corresponding employed bee memorizes this new solution and forgets the old one.

Bayesian Decision

To complete the last step of decision, ρ has to be estimated from the observations $\{\xi_i, i = 1, 2, \ldots, n\}$ and $\{\eta_i, i = 1, 2, \ldots, n\}$, and then compared with $\rho'(c)$ according to (4.21). We assume there is a module named *Predictor* that fulfills this last step. In our implementation, the *Predictor* calculates the average value of ρ and outputs the comparison of this average value and $\rho'(c)$. However, this implementation is only a simple one, and it can be replaced with other advanced implementation. Thus, the *Predictor* module can be treated as a black box [Wolski *et al.*, 2008].

Although the observations are known, the distribution of ρ is unknown in most scenarios. Thus, to estimate ρ analytically is not practical. Besides, the network is always dynamic, which is not fully understood yet. Therefore, to predict future network state based on historical observation is usually done empirically. From this perspective, the comparison result from black box *Predictor* cannot be fully trusted. Thus, we use another module named *Bayesian Decision* to mitigate the errors of *Predictor* and to make the final decision for (4.21).

For simplicity, let T be $\rho'(c)$. And we assume $\omega_{offload}$ denotes the comparison result $\rho > \rho'(c)$, and ω_{local} denotes $\rho \leq \rho'(c)$. The Bayesian risk of decision to offload is the expected cost as a function of the posterior distribution associated with

Algorithm 2 Application partition algorithm details.

```
1: Initialize employed bees randomly $\mathbf{x}_i$: the $i^{\text{th}}$ employed bee
2: $cycle \leftarrow 1$
3: repeat
4:     for each employed bee $\mathbf{x}_i$ do
5:         Apply $Flip$ local search and find $\mathbf{x}_{flip}$
6:         Apply $Swap$ local search and find $\mathbf{x}_{swap}$
7:         if $\rho'(\mathbf{x}_i) > \min\{\rho'(\mathbf{x}_i), \rho'(\mathbf{x}_{flip}), \rho'(\mathbf{x}_{swap})\}$ then
8:             $\mathbf{x}_i \leftarrow \arg\min\{\rho'(\mathbf{x}_i), \rho'(\mathbf{x}_{flip}), \rho'(\mathbf{x}_{swap})\}$
9:         end if
10:    end for
11:    Determine probabilities $(p_i)$ by (4.22)
12:    $M_i \leftarrow p_i M$: number of onlooker bees sent to the $i^{\text{th}}$ food source
13:    $\mathbf{y}_{ij} \leftarrow \mathbf{x}_i$ $(j = 1, 2, \ldots, M_i)$: the $j^{\text{th}}$ onlooker bee of the $i^{\text{th}}$ food source
14:    for each onlooker bee $\mathbf{y}_{ij}$ do
15:        Apply $Flip$ local search and find $\mathbf{y}_{flip}$
16:        Apply $Swap$ local search and find $\mathbf{y}_{swap}$
17:        if $\rho'(\mathbf{y}_{ij}) > \min\{\rho'(\mathbf{y}_{ij}), \rho'(\mathbf{y}_{flip}), \rho'(\mathbf{y}_{swap})\}$ then
18:            $\mathbf{y}_{ij} \leftarrow \arg\min\{\rho'(\mathbf{y}_{ij}), \rho'(\mathbf{y}_{flip}), \rho'(\mathbf{y}_{swap})\}$
19:        end if
20:    end for
21:    for each employed bee $\mathbf{x}_i$ do
22:        if $\rho'(\mathbf{x}_i) > \min_{j=1,2,\ldots,M_i}\{\rho'(\mathbf{y}_{ij})\}$ then
23:            $\mathbf{x}_i \leftarrow \arg\min_{j=1,2,\ldots,M_i}\{\rho'(\mathbf{y}_{ij})\}$
24:        end if
25:    end for
26:    Generate scout bee $\mathbf{z}$ randomly
27:    if $\max_{i=1,2,\ldots,N}\{\rho'(\mathbf{x}_i)\} > \rho'(\mathbf{z})$ then
28:        $\arg\max_{i=1,2,\ldots,N}\{\rho'(\mathbf{x}_i)\} \leftarrow \mathbf{z}$
29:    end if
30:    if $\rho'(\mathbf{x}_{best}) > \min_{i=1,2,\ldots,N}\{\rho'(\mathbf{x}_i)\}$ then
31:        $\mathbf{x}_{best} \leftarrow \arg\min_{i=1,2,\ldots,N}\{\rho'(\mathbf{x}_i)\}$
32:    end if
33:    $cycle \leftarrow cycle + 1$
34: until $cycle = MCN$
```

offloading:

$$\begin{aligned} R_{offload} &= \int_0^T \mathrm{C}_{offload}(\rho) f(\rho|\omega_{offload}) \mathrm{d}\rho \\ &= \mathrm{C}_{offload}(\rho) \mid_{\rho=\int_0^T f(\rho|\omega_{offload})\mathrm{d}\rho} \end{aligned} \tag{4.23}$$

where loss function $\mathrm{C}_{offload}(\rho)$ can be either time expense (4.11) or energy expense (4.16) according to user preference. And $f(\rho|\omega_{offload})$ is the probability of observing a network availability value ρ given the *Predictor*'s output $\omega_{offload}$. When ρ is greater than T, offloading does not pay penalty, thus the integral range is $[0, T]$. Similarly, the risk of decision to execute locally is:

$$\begin{aligned} R_{local} &= \int_T^1 \mathrm{C}_{local} f(\rho|\omega_{local}) \mathrm{d}\rho \\ &= \mathrm{C}_{local} \int_T^1 f(\rho|\omega_{local}) \mathrm{d}\rho \end{aligned} \tag{4.24}$$

where loss function C_{local} is constant value calculated by either (4.1) or (4.10) according to the same user preference used for (4.23). We may pay penalty in both offloading or local execution scenarios, which is the risk behind the above two equations. When offloading is applied, the execution time or energy consumption may exceed the cost of local execution according to (4.11) and (4.16). Similarly, if local execution is chosen, the benefit may be lost when (4.21) is satisfied.

To calculate (4.23), the posterior Probability Distribution Function (PDF) $f(\rho|\omega_{offload})$ can be calculated by Bayes theorem:

$$f(\rho|\omega_{offload}) = \frac{f(\omega_{offload}|\rho) f(\rho)}{f(\omega_{offload})} \tag{4.25}$$

where $f(\rho)$ is the prior PDF of availability ρ, $f(\omega_{offload}|\rho)$ is the conditional PDF of event $\omega_{offload}$ given a availability ρ. $f(\rho|\omega_{local})$ is calculated in the same way for (4.24).

We maintain a histogram for $f(\rho)$. The value range $[0, 1]$ is divided into many slots, each of which is indexed by a ρ value, and each of which is corresponding to a bin in histogram. When a particular ρ value is observed, it falls into one bin. The bins accumulate the observation count. To calculate $f(\omega_{offload}|\rho)$ and $f(\omega_{local}|\rho)$, We maintain two counters for each bin in the histogram. One counter is for counting event $\omega_{offload}$ and the other is for counting ω_{local}. The events $\omega_{offload}$ and ω_{local} are counted according to their corresponding bin indexed by availability ρ. There are another two global counters that count the events $\omega_{offload}$ and ω_{local} no matter what availability value ρ is associated. These two counters are used to calculate $f(\omega_{offload})$ and $f(\omega_{local})$.

For example, assume the histogram has ten bins, which means the range $[0, 1]$ is divided into ten slots. The first bin is corresponding to range $[0, 0.1]$; the second bin is corresponding to range $[0.1, 0.2]$, and so forth. When we observe an availability value $\rho = 0.47$, we add 1 to the fifth bin $[0.4, 0.5]$. Assume the corresponding event of this observation is ω_{local}, we add 1 to the counter of ω_{local} associated with the fifth bin, and add 1 to the global counter of ω_{local} as well.

Offloading Decision Module

Based on the above discussions, the offloading decision module is depicted in Figure 4.3. The offloading decision module maintains a buffer that stores network states duration sequences. The buffer is constructed as a table that has two columns ξ and η. Every row is an observation of cycle of alternating renewal process. The rows are sorted with the latest observation on the top and oldest observation at bottom. When the table is full, the oldest tuples are removed from the bottom. When application information is provided, *Application Partition* executes the algorithm in Algorithm 1 and Algorithm 2 and outputs threshold T and configuration c. *Predictor* reads buffer and T, then gives availability prediction. Then *Bayesian Decision* finally makes the decision. The final output of the offloading decision module is either a configuration

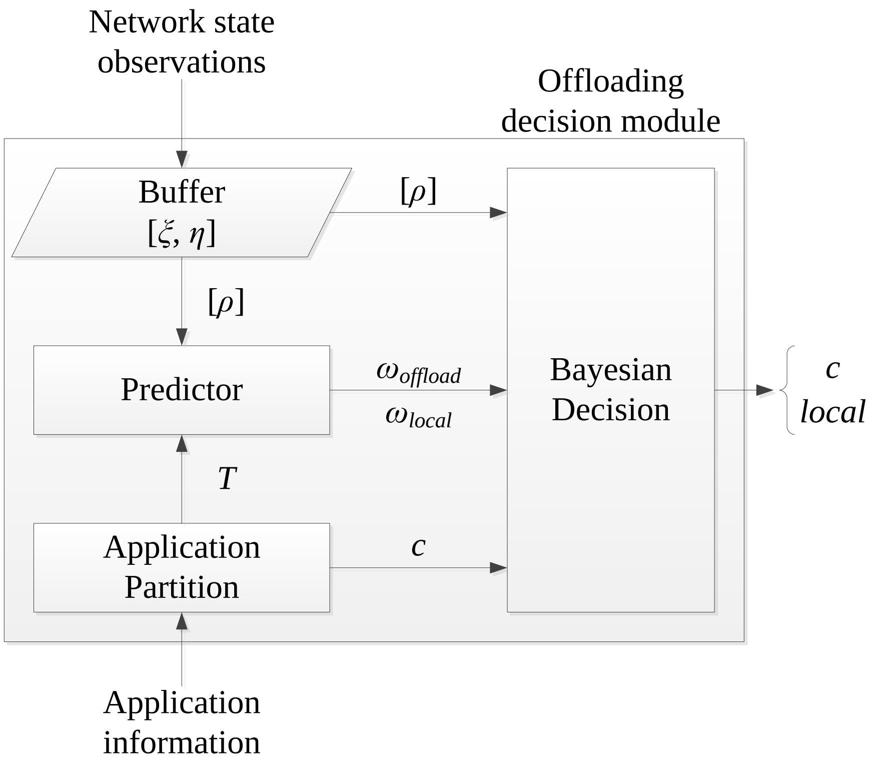


Figure 4.3: Offloading Decision Module for Unstable Network Scenario.

or a flag indicating local execution.

4.1.3 Simulation and Analysis

In this section, we evaluate ABC based partition algorithm including algorithm tuning and impact of different mobile application and cloud. We evaluate our model and algorithms in MATLAB.

We generate two hundred random application graphs as base evaluation data set. The default parameter settings are shown in Table 4.1. We use a set of typical energy parameters K for a phone according to [Wang *et al.*, 2013]. The cloud-phone processing ability ratio q varies in large range from previous work [Wolski *et al.*, 2008][Xian *et al.*, 2007]. We pick a medium value from the possible range as default value and evaluate its impact to algorithm in section 4.1.3. We evaluate the algorithm in three aspects: bee colony, different applications and cloud-phone relation.

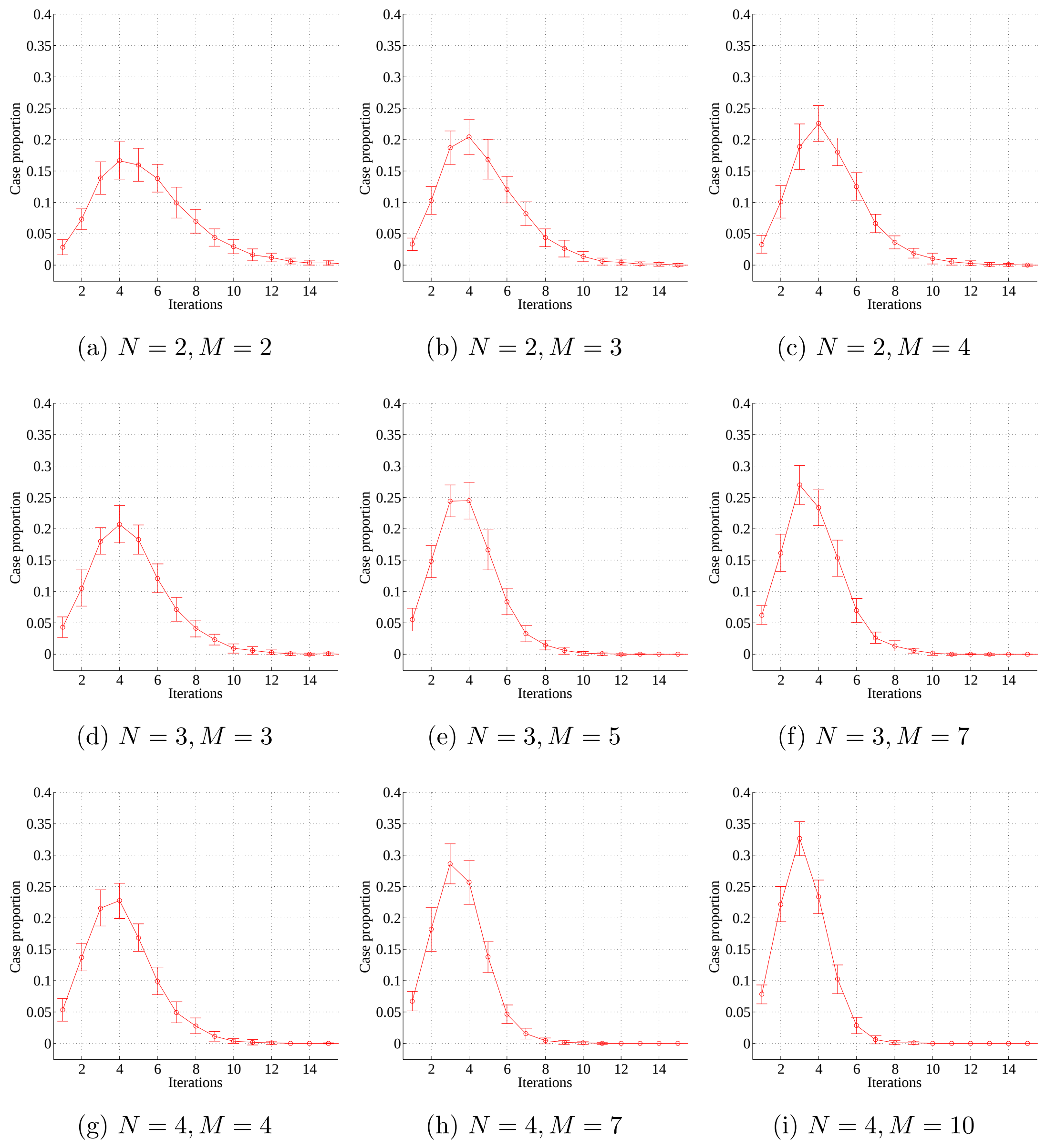
Table 4.1: Default Parameter Setting for Offloading Decision Evaluation Under Unstable Network Scenario.

Parameter		Default value
Application	m	10
	n	8
Cloud [Wolski *et al.*, 2008][Xian *et al.*, 2007]	q	30
Phone [Wang *et al.*, 2013]	K_{CPU}^{sta}	2.5
	K_{CPU}^{dyn}	5
	K_{RF}^{sta}	1.25
	K_{RF}^{dyn}	1.25
Algorithm	N	3
	M	5

Bee Colony and Algorithm Tuning

This experiment is based on two hundred random application graphs ($\mathbf{t}$ and $\mathbf{D}$). These application graphs are randomly generated, and at least one configuration of each graph is guaranteed to obtain both time and energy benefit in ideal network environment. We evaluate the proposed algorithm performance of difference bee colony size. The results are shown in Figure 4.4. The x-axis represents how many iterations that the algorithm needs to reach the $\mathbf{x}_{best}$, and y-axis represents how many cases reach the solution of corresponding iterations. From the figure, we may draw following guides for algorithm tuning:

- Increasing onlooker bee number, algorithm shows the better convergent rate. For the same employed bee number (N), the more onlooker bees there are, the less iterations are required to reach the optimal solution obviously in all three situations ($N = 2, 3, 4$).



(a) $N = 2, M = 2$ (b) $N = 2, M = 3$ (c) $N = 2, M = 4$

(d) $N = 3, M = 3$ (e) $N = 3, M = 5$ (f) $N = 3, M = 7$

(g) $N = 4, M = 4$ (h) $N = 4, M = 7$ (i) $N = 4, M = 10$

Figure 4.4: Partition Performance of Different Bee Colonies.

- Increasing employed bee number improves convergent rate, but the improvement is not obvious compared to increasing onlooker bee number. For the same onlooker bees of $M = 3$ (Figure 4.4b, Figure 4.4d), $M = 4$ (Figure 4.4c, Figure 4.4g) and $M = 7$ (Figure 4.4f, Figure 4.4h), the cases that have more employed bees have slightly performance improvement, which is not as obvious as the improvement given by increasing onlooker bees. For $M = 4$ figures, more than 0.05 cases reach the optimal solution at iteration number 7 in Figure 4.4c while there are only 0.05 cases reach solution at iteration number 7, which means more cases reach solution in less than 7 iterations, in Figure 4.4g. Similar phenomena is found for $M = 3$ and $M = 7$ figures.

- For the same total bee number of employed bees and onlooker bees, the algorithm prefers more onlooker bees slightly. For the same total bees of $N + M = 6$ (Figure 4.4c, Figure 4.4d) and $N + M = 8$ (Figure 4.4e, Figure 4.4g), we can see that the overall performance are almost the same. But the iterations to get optimal solutions in the cases that have more onlooker bees are slightly concentrated on some iteration numbers. For $N + M = 6$, the iterations in Figure 4.4c are concentrated demonstrated by higher summit at iteration 4, while it is diversely distributed from 1 to 8 in Figure 4.4d. Similar situation occurs for $N + M = 8$.

Application Impact

To evaluate the algorithm performance for difference applications, three experiments are done for component number, unmovable component proportion and computation-communication ratio separately. The experiment result for difference component number is shown in Figure 4.5. For the same bee colony, the iterations to find the solution increases along with the component number. For the large applications that have many components, the algorithm may use large bee colony to assure small iterations.

We evaluate the impact of unmoveable component proportion of application in the second experiment and the result is shown in Figure 4.6. From the figure, we find that more iterations are required to get the solution when the movable component number increases. The trend is very like that in Figure 4.5, which implies that the unmoveable component number does not play much role in the algorithm performance. This is because the algorithm always consider the movable components and ignore the unmovable component when generating new solutions in each cycle. The total component size increase in Figure 4.5 leads to the increase of movable component number, which is like what this experiment does in Figure 4.6. Besides, we also found in this experiment that higher movable proportion results in the robust solution that can work under low network availability $\rho'(c)$ situations. This is because the high movable proportion provides more candidate partition options so that more robust solution may be achieved.

We generate another two sets of application graph data of different computation load. The data set used in previous experiments are used as reference data set. Then we adjust the computation task to half and double of the reference data set in the application graph generation process. The network task remains the same, thus the computation-communication ratio is adjusted to half and double in new data sets. The experiment results for these three data set are shown in Figure 4.7. From the figure, we can see that the iterations, distributed at 2,3,4,5 and 6, are almost the same, which implies the computation proportion of the given task does not influence the algorithm performance. This is because the computation proportion in the task influence the time benefit and energy benefit in the same direction. In the experiment, we also find

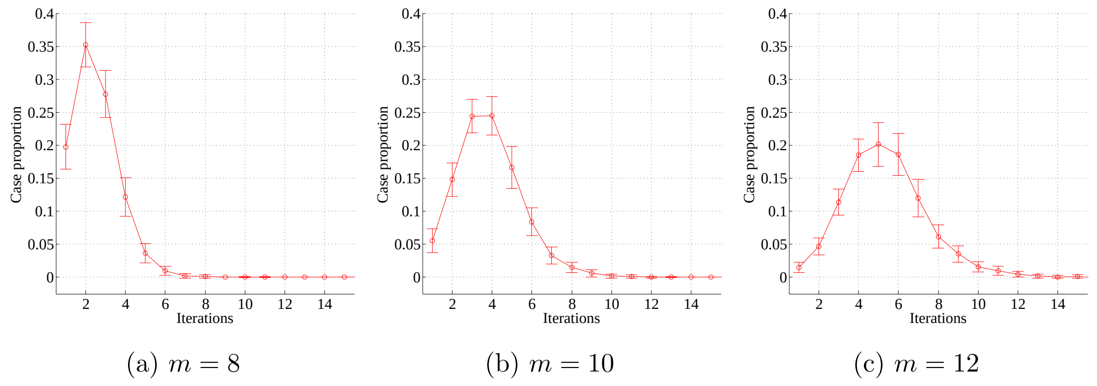

(a) $m = 8$ (b) $m = 10$ (c) $m = 12$

Figure 4.5: Partition Performance of Different Component Numbers.

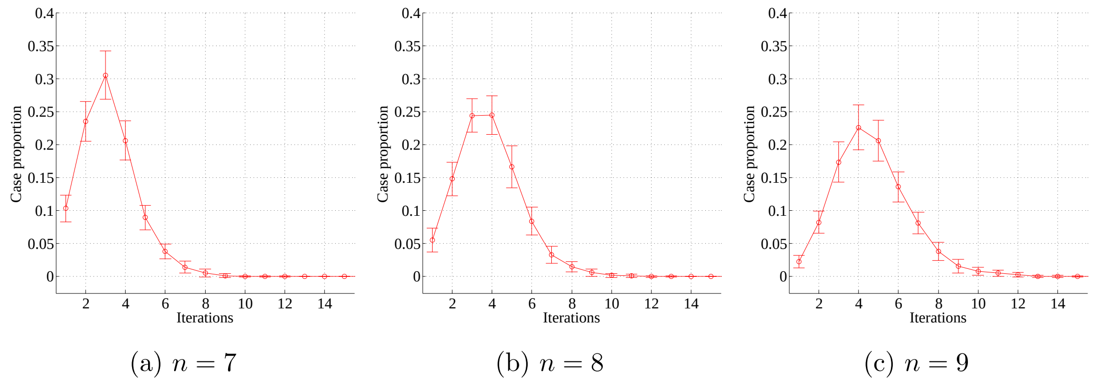
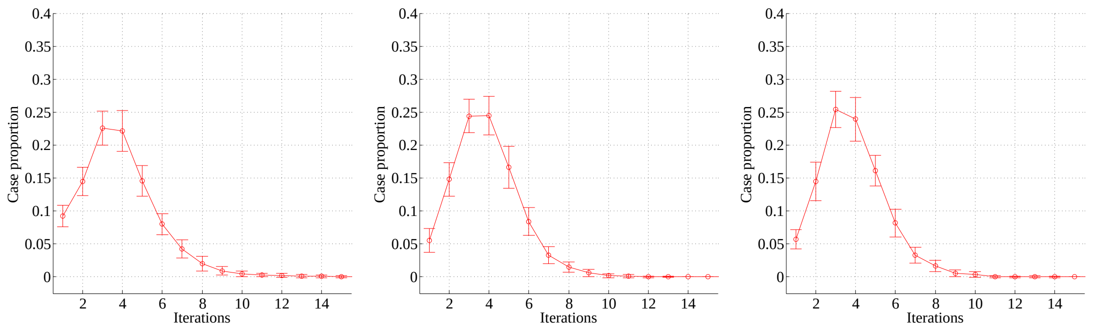

(a) $n = 7$ (b) $n = 8$ (c) $n = 9$

Figure 4.6: Partition Performance of Different Unmovable Component Proportions.

that the computation proportion impacts the $\rho'(c)$, because the more computation proportion leads to more offloading benefit and the possible solution is more resistent to network unavailability.

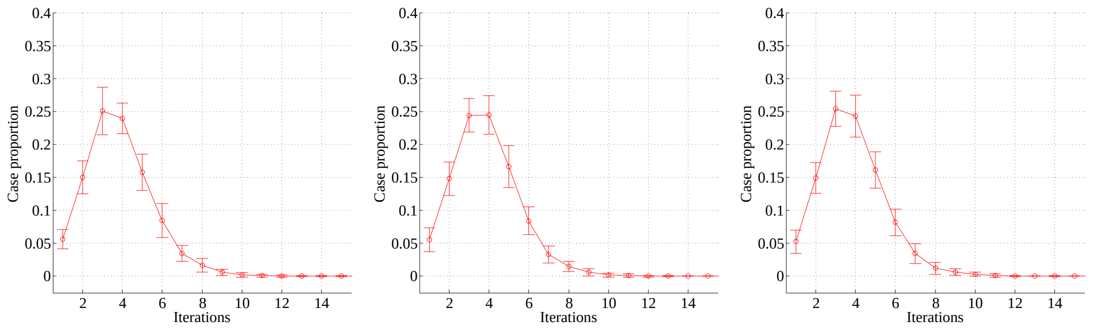
Cloud Impact

We evaluate the algorithm performance under different cloud speedup ratios shown in Figure 4.8. The figure shows that the iteration number does not depend on the cloud processing capability. The cloud processing capability influences the execution time considered in algorithm, which is similar to the computation proportion impact. And similarly, the higher cloud processing capability results in more robust partition configuration.



(a) half computation task (b) reference computation task (c) double computation task

Figure 4.7: Partition Performance of Different Computation Tasks.



(a) $q = 5$ (b) $q = 10$ (c) $q = 20$

Figure 4.8: Partition Performance of Different Cloud Speedup Ratios.

4.2 Multiple Surrogates and Multiple Objectives

The presented multi-factor multi-site risk-based offloading model abstracts the offloading impact factors into two categories: offloading benefit and offloading risk. The final offloading decision is made based on the aggregated and normalized benefit and risk evaluations. This presented model is generic and expendable in that the number of factors that are considered in the offloading decision processes varies. Specifically, we present four offloading impact factors to show how the model works, and it can cover more factors to make offloading decision more comprehensive. The offloading benefits include delay reduction and energy saving, and offloading risks factors include privacy breach and reliability of the offloading targeting nodes. The overall offloading benefits and risks are aggregated based on the mobile cloud users' preference. Finally, we design an ant-based algorithm [Taillard and Gambardella, 1997] to compute the optimal application partition strategy. A reference implementation is also presented following the approach description. In summary, the contributions of presented research are highlighted as follows:

- *Multi-site offloading*: The proposed offloading approach picks one or multiple surrogates from a set of candidate sites (here we use *node* and *site* interchangeably), i.e., there could be multiple offloading destinations, where both cloud VMs and mobile nodes can serve as a destination. The consideration of multiple surrogates results in a more flexible offloading model for running mobile cloud applications.

- *Risk-based multi-factor decision*: The proposed approach takes two risk factors into offloading decision making process. The privacy risk and reliability risks have not been studied in previous work, where they are common issues in mobile cloud computing application scenarios. The offloading decision is made by comparing the aggregated risk and potential offloading benefit. The proposed decision approach is not limited to only two types of risk or benefits. The solution can be easily extended to many factors as long as the factors result in either benefits or risks.

- *Application offloading and composition*: The proposed approach can be used to model both application offloading and composition. In the above presented example, the vehicles provide their video capturing function as an application component (or function). This video capturing function can be composed with other application components to composite a new mobile cloud application for the requester.

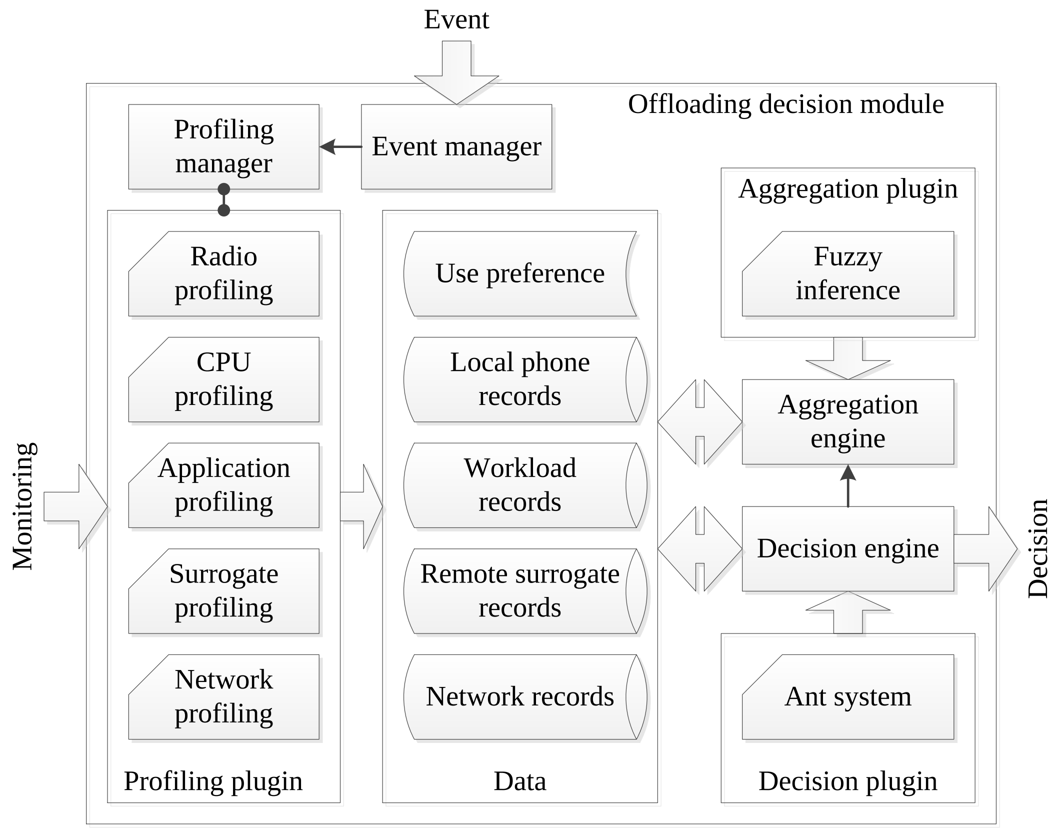


Figure 4.9: Offloading Decision Module Architecture for Multi-objective Scenario.

4.2.1 System and Models

In our system model, each Application Component (AC) provisioning node is considered as an application *site* that runs one or multiple ACs based on the Java framework [Wu *et al.*, 2015]. The offloading decision module shown in Figure 4.9. It has two tiers: monitoring and decision making, which are connected by a data pool. The monitoring tier includes *event manager*, *profiling manager*, and *profiling plugins*. *Event manager* monitors the events that trigger the offloading decision making process. When an event is received, the *event manager* triggers the *profiling manager* to distribute 'output data' commands to all *profiling plugins*. The *profiling plugins* are used to monitor and cache environment data. When they receive 'output data' commands, they flush the collected data to *data pool* and trigger the *aggregation engine* to preprocess data. The decision making tier includes several components, i.e., *aggregation engine*, *aggregation plugin*, *decision engine*, and *decision plugin*. *Aggregation engine* and *decision engine* control the work flow, while *aggregation plugin* and *decision plugin* actually do the processing work. The *aggregation engine* picks data from the *data pool* and outputs the aggregated data back to the pool; then the *decision engine* runs the decision making process by first reading from the *data pool* and then outputting a final decision. In the decision making process, the *decision engine* may call *aggregation engine* to aggregate data. This module is one of components that compose the complete application. It usually runs on the original site, however, this module may be offloaded as well to cloud or other mobile devices.

The proposed offloading decision module can work in an online manner. The modules involved in online processing include the *event manager* and *profiling plugins*. The online processing is implemented based on event driven. There are two event sources: significant profiling data change and a regular timer. The profiling plugin daemons monitor changes of resource provisioning or environment situation fluctuations, and then send trigger events to the *event manager*; the timer sends out the trigger event periodically.

Application Partition

The parties involved in the offloading process include the ACs and the surrogates. The offloading solution is actually a mapping from ACs to surrogates.

Application graph The modern mobile application is usually guided by component oriented design. Components provide functionality via interfaces and may, conversely, consume functionality provided by other components [Wu *et al.*, 2013]. The application can be presented as a graph $G_{app} = (C, E)$ where the vertex is compo-

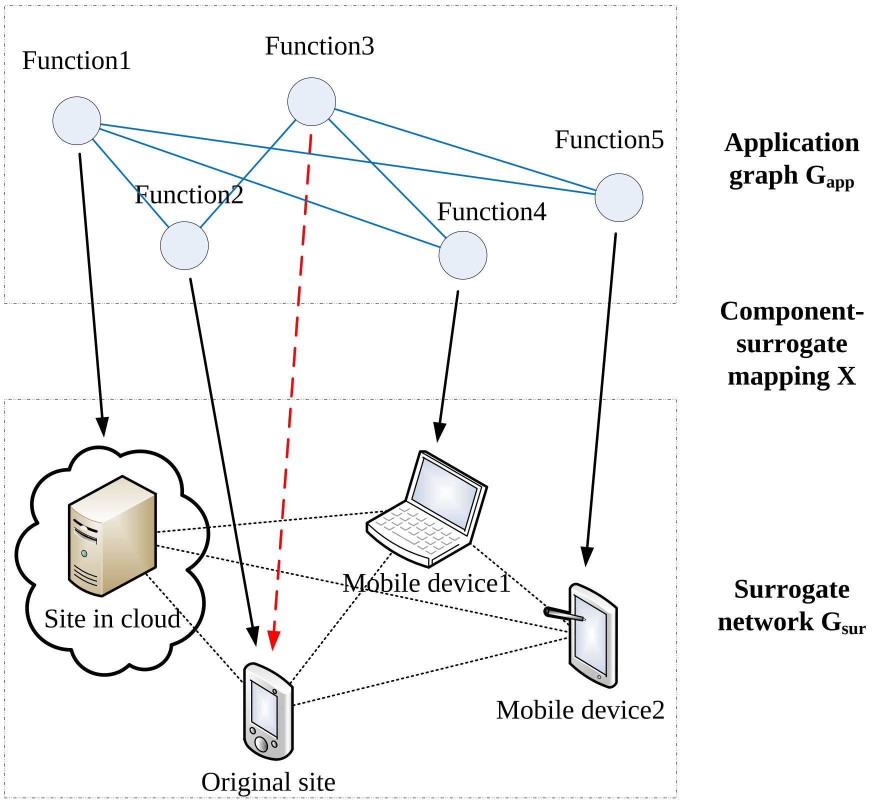


Figure 4.10: Component-surrogate Mapping.

nent and the edge is interaction between components or the data transfer or data dependency between components.

To be extensible, the application graph is not limited to present only one application. The application graph can be extended to a big graph that contains several applications and even the connections between applications.

Surrogate network In a multi-site service composition scenario, some computation workload is offloaded from original mobile device and distributed onto the picked surrogate sites from candidate sites. The original site and its candidate sites form an egocentric network $G_{sur} = (S, L)$, where the node is site and the link is network connection between sites.

Component-surrogate mapping The service composition mapping is a mapping from application graph G_{app} to a surrogate network G_{sur}, where vertexes are mapped to nodes and the edges are mapped to links correspondingly, as shown in Fig. 4.10. Let matrix X present this mapping, whose element x_{ij} is set to either 1 when component i is assigned to site j or 0, otherwise. Matrix X has following properties:

1. Let m be the component number $m = |C|$ and n be the candidate site number $n = |S|$. Then, X is $X_{m\times n}$. The relation of m and n is not strict, which means $m > n$, $m < n$ and $m = n$ are all valid.

2. The mapping is as well not strict, which means several components may be mapped to the same surrogate site and some surrogate site may not host any components. However, each component in the application is assigned to exact one site, thus, there is one and only one 1 in each row of X: $\sum_{j=1}^{n} x_{ij} = 1$ for every row i.

3. In a mobile application, some components have to be assigned to particular sites due to application requirements. For example, human machine interaction components and sensor components have to be put on the original mobile device because they use mobile device hardware that is not available on surrogate sites. This requirement enforces that positions of some 1's in some rows of X are predefined and cannot be moved. These rows are put at the bottom of the X. Except these rows, the rest of X is the effective matrix $X_{\tilde{m}\times n}$, which corresponds to the movable components.

Based on X, four more mappings can be defined for easy expression in following sections. Mapping $f_{C\to S}$ implements the similar function as X, which maps component i to site j: $f_{C\to S}(i) = j, \forall x_{ij} = 1, x_{ij} \in X$. Mapping $f_{S\to C}$ is the reverse mapping of $f_{C\to S}$, which given site j outputs a set of components $\{i_1, i_2, \ldots, i_k\}$ coded as a vector $\mathbf{i}$. Besides, mappings can also be defined between E and L. Mapping $f_{E\to L}$ maps edge $e = (i_1, i_2)$ to link $l = (f_{C\to S}(i_1), f_{C\to S}(i_2))$. And similarly to $f_{S\to C}$, mapping $f_{L\to E}$ maps a link to a vector of edges.

Offloading Benefits

Based on a certain risk, two primary offloading goals are considered: one is to reduce the overall application execution time and the other is to save energy on mobile device.

Execution Time According to [Wang *et al.*, 2013], at any time, the computation load for every component and the volume of data exchange for every edge in application graph is known based on a given task or is predictable based on the application behavior and user historical behavior. The workload is distributed on the application graph, which makes G_{app} into a weighted graph. The computation load is presented as a weight value on each vertex: the component c is labeled with computation load w_c and vector $\mathbf{w}$ is computation load of all vertexes. The data exchange amount is presented as a weight value on an edge: the edge $e = (c_1, c_2)$ from c_1 to c_2 is labeled with data transfer load $f_{c_1 c_2}$ and matrix $F_{m\times m}$ is data exchange load of all edges. The diagonal of F are all 0's because the component's interaction with itself does not count for inter-component data exchange load; and $f_e = 0$ if $e \in E$ is false.

Meanwhile, the surrogates' capability is measurable, so the egocentric network G_{sur} is transformed into a weighted graph as well. The available computation capability on a candidate site s is labeled as a vertex weight u_s and vector $\mathbf{u}$ is weights of all sites. The network throughput on link $l = (s_1, s_2)$ from s_1 to s_2 is labeled as link weight $d_{s_1 s_2}$ and matrix $D_{n\times n}$ is weights of all links. The diagonal of D are large values because data exchange in the same site can be considered negligible.

In this article, we define the operator $.*$ as array inner multiplication that multiplies arrays or matrices in element-by-element way, which is different from matrix multiplication. Let $\tilde{\mathbf{u}}$ be the vector that satisfies $\mathbf{u}. * \tilde{\mathbf{u}} = \mathbf{1}$ where $\mathbf{1}$ is the vector whose elements are all 1's. Then the upper bound of total time spent for computation load is sum of workload on every site over its processing capability:

$$t_c = \mathbf{w}^T X \tilde{\mathbf{u}}. \tag{4.26}$$

Similarly, let $\tilde{D}_{n\times n}$ be the matrix that satisfies $D.*\tilde{D} = \mathbf{1}_{n\times n}$ where $\mathbf{1}_{n\times n}$ is the matrix whose elements are all 1. The transformation $X^T F X$ redistributes the communication load of $F_{m\times m}$ into a $n \times n$ matrix where element positions are corresponding to $\tilde{D}_{n\times n}$. The upper bound of total time spent on networks for data exchange load is the sum of workload on every link over its throughput:

$$t_n = tr(X^T F X \tilde{D}^T), \tag{4.27}$$

where the $tr()$ function calculates matrix trace $tr(A_{n\times n}) = \sum_{i=1}^{n} a_{ii}$. So the upper bound of total time is the sum of the computation time and the communication time:

$$t = t_c + t_n. \tag{4.28}$$

Energy Consumption Energy consumption on mobile devices can be categorized according to hardware modules. The major categories are CPU, radio module including Wi-Fi and Cellular, display, audio device, GPS module and vibration motor [Shin *et al.*, 2013][Yoon *et al.*, 2012][Zhang *et al.*, 2010a][Jung *et al.*, 2012]. The components that involve modules except CPU and radio are hardware dependent components that have to run on the original mobile device and that are not considered for offloading. Both CPU and radio power can be modeled as a linear model that consists of two parts: dynamic consumption when hardware module is active and base consumption [Wang *et al.*, 2013][McCullough *et al.*, 2011]. The dynamic part of CPU power is proportional to utilized processing capability according to [Jung *et al.*, 2012][Zhang *et al.*, 2010a][Mittal *et al.*, 2012][Shin *et al.*, 2013]:

$$P^{CPU} = \mathbf{1}^T X(\mathbf{u}. * \mathbf{c}^{CPU}), \tag{4.29}$$

where $\mathbf{c}^{CPU}$ is the coefficient vector for all sites; and the coefficients of non-mobile sites in $\mathbf{c}^{CPU}$ are 0's. Let's code E as $\mathbf{e}$, then the dynamic part of radio module power is proportional to the outgoing packet rate [Jung *et al.*, 2012][Zhang *et al.*, 2010a]:

$$D' = D. * (\mathbf{1}_{n \times n} - I). * (\mathbf{c}^{radio}\mathbf{1}^T), \tag{4.30}$$

$$P^{radio} = \sum_{\forall l \in f_{E \to L}(\mathbf{e})} d'_l, \tag{4.31}$$

where $\mathbf{c}^{radio}$ is coefficient vector for all sites, I is identity matrix and d'_l is the element of D' corresponding to link l. The coefficients of non-mobile sites in $\mathbf{c}^{radio}$ are 0's. Let P^{CPU}_{idle} be the static part of CPU power and P^{radio}_{base} be static power of radio module. Then the total power is the sum of the above four parts:

$$P = P^{CPU} + P^{CPU}_{idle} + P^{radio} + P^{radio}_{base}. \tag{4.32}$$

Offloading Risks

The offloading process is always involved with some risks. Risk (r_e) is defined as a product of probability (p_e) of an occurring event (e) and its potential impact (q_e) or consequence $r_e = p_e q_e$ [Saripalli and Walters, 2010]. Two major types of events in mobile cloud offloading process are: information leakage and surrogate reliability, and these events are assumed to be independent.

Privacy and Trust Information leakage may happen in transportation process in network and/or in computation process on sites. Let's code C as $\mathbf{c}$ and E as $\mathbf{e}$, then combine $f_{C\to S}(\mathbf{c})$ and $f_{E\to L}(\mathbf{e})$ as a vector $\mathbf{v}$ of elements that may leak information. Corresponding to this vector, two state vectors $\mathbf{s}$ and $\mathbf{s}'$ are defined of one unique surrogate network state. Vector $\mathbf{s}$ records what elements leak information and vector $\mathbf{s}'$ records links on which data is exposed. Both $\mathbf{s}$ and $\mathbf{s}'$ are constructed initially as all 0's and then adjusted according to:

- For $\mathbf{s}$, if v_i leaks information, then $s_i = 1$.
- For $\mathbf{s}'$, two situations are considered. First, if a link $v_i \in f_{E\to L}(\mathbf{e})$ leaks information, then $s'_i = 1$. Second, if a site is compromised, then all data transferred on incident links are exposed. Let $v_i \in f_{E\to L}(\mathbf{e})$ and $v_i = (v_j, v_k)$, and either v_j or v_k leaks information, then $s'_i = 1$.

Let function $p_1(v_i)$ be the probability of leakage occurring for element v_i. The implementation of this probability is discussed in section 4.2.3. The probability of state $\mathbf{s}$ is the probability product of all independent leakage events:

$$p_{1\mathbf{s}} = \prod_{\forall s_i=1} p_1(v_i) \prod_{\forall s_i=0} (1 - p_1(v_i)). \tag{4.33}$$

The amount of exposed data is considered as the information leakage consequence. The impact of state $\mathbf{s}'$ is data sum of all unique leakage events:

$$q_{1\mathbf{s}'} = \sum_{\forall s'_i=1} \{f_{\tilde{\mathbf{e}}}\}, \tag{4.34}$$

where $f_{\tilde{\mathbf{e}}}$ are values picked from F corresponding to edges in $\tilde{\mathbf{e}} = f_{L\to E}(v_i)$; and $\{f_{\tilde{\mathbf{e}}}\}$ is set of data loss in all unique leakage events.

All valid $\mathbf{s}(\mathbf{s}')$ vectors form a state space H and $\sum_{\mathbf{s}\in H} p_{1\mathbf{s}} = 1$. The risk can be computed based on the expected data loss:

$$r_1 = \sum_{\mathbf{s}(\mathbf{s}')\in H} p_{1\mathbf{s}} q_{1\mathbf{s}'}. \tag{4.35}$$

Reliability The surrogates may be unavailable due to device mobility or failures in network or on sites. Similarly, vector $\mathbf{v}$ is defined of elements that may be unavailable and state vectors $\mathbf{s}$ and $\mathbf{s}'$ are defined as well [Jereb, 1998][Levendovszky *et al.*, 2002][Booker *et al.*, 2008]. Vector $\mathbf{s}$ records what elements are unavailable and vector $\mathbf{s}'$ records nodes on which workload is lost. The $\mathbf{s}$ and $\mathbf{s}'$ are initialized as all 0's and the following steps are executed:

- For $\mathbf{s}$, if v_i is unavailable, then $s_i = 1$.
- For $\mathbf{s}'$, two situations are considered. First, if a site $v_i \in f_{C\to S}(\mathbf{c})$ is unavailable, then $s'_i = 1$. Second, if a link fails, then the node that accepts the data on that link will fail due to lack of input. Let $v_i \in f_{E\to L}(\mathbf{e})$ and $v_i = (v_j, v_k)$ is unavailable, then $s'_k = 1$.

Let function $p_2(v_i)$ be probability of unavailability occurring for element v_i. The implementation of this probability is discussed in section 4.2.3. The probability of state $\mathbf{s}$ is probability product of all independent unavailability events:

$$p_{2\mathbf{s}} = \prod_{\forall s_i=1} p_2(v_i) \prod_{\forall s_i=0} (1 - p_2(v_i)). \tag{4.36}$$

The loss of computation is considered as the consequence of the unavailability of surrogates. The impact of state $\mathbf{s}'$ is sum of computation loss in all unique unavailability events:

$$q_{2\mathbf{s}'} = \sum_{\forall s'_i=1} \{w_{\tilde{\mathbf{c}}}\}, \tag{4.37}$$

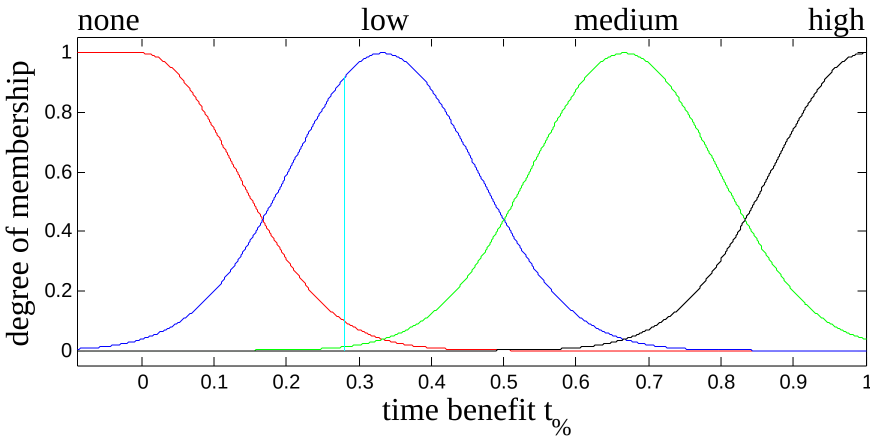


Figure 4.11: Time Benefit Member Functions.

where $w_{\tilde{\mathbf{c}}}$ are values picked from $\mathbf{w}$ corresponding to components in $\tilde{\mathbf{c}} = f_{S \to C}(v_i)$; and $\{w_{\tilde{\mathbf{c}}}\}$ is a set of computation loss of all unavailability events.

The expected computation loss or risk r_2 is calculated in the same way with (4.35):

$$r_2 = \sum_{\mathbf{s}(\mathbf{s}') \in H} p_{2\mathbf{s}} q_{2\mathbf{s}'}. \tag{4.38}$$

User Preference

The benefit originates from several sources. However, we only use time and energy benefits to simplify the presentation of multi-factor measurement. The sources of benefits are usually presented in different format and thus they cannot be simply processed together. We classify benefits from each source into several levels, such as *none*, *low*, *medium*, and *high*, so that each type of benefits can be processed together. To obtain the overall offloading benefit, user preference is used to aggregate time benefit and energy benefit. A user usually does not specify distinct preference between time benefit and energy benefit. Here we use a fuzzy-based approach [Ni *et al.*, 2010], where the overall benefit can be calculated by fuzzy inference in three steps: fuzzification, inference and defuzzification.

Table 4.2: Benefit Inference Rules

Weight	Time benefit	Energy benefit	Final benefit
...	...	...	...
0.85	low	medium	medium
...	...	...	...

1. The time benefit is firstly normalized by $t_{\%} = (t_{orig} - t)/t_{orig}$ where t_{orig} is execution time when no offloading is done, which is equivalent to mapping all components to only original device in X. The membership degree of $t_{\%}$ to a specific benefit level is determined by membership function. For example, the membership degrees of $t_{\%} = 0.28$ are: $\mathrm{mf}_{none}^{time}(0.28) = 0.0983$ (*none*), where $\mathrm{mf}_{none}^{time}$ is member function of time benefit for level '*none*', and similarly 0.9193 (*low*), 0.0120 (*medium*), and 0 (*high*) according to membership functions shown in Figure 4.11. Similarly, the energy benefit is fuzzified as well.

2. The firing degree of a rule is calculated based on the chosen conjunction operation and membership degrees of both time and energy benefit. For example, the chosen operation is product t-norm $T_p(a, b, \ldots) = ab\ldots$, and the membership degree of energy benefit for '*medium*' is 0.5642. Then, the firing degree is calculated by $T_p(0.9193, 0.5642) = 0.9193 \times 0.5642 = 0.5187$ for the rule shown in Table 4.2.

   The final benefit of a rule is estimated based on the conjunction of its member function, its firing degree, and its weight. For instance, the final benefit for the rule in Table 4.2 is: $T_p(0.5187, 0.85, \mathrm{mf}_{medium}^{final}) = 0.5187 \times 0.85 \times \mathrm{mf}_{medium}^{final} = 0.4409 \times \mathrm{mf}_{medium}^{final}$ where $\mathrm{mf}_{medium}^{final}$ is member function of the final benefit for '*medium*'.

The preference given by a rational user is expected to be monotonic to both time and energy benefit. The final benefit increases when both benefits increase or when either benefit increases and the other stays. And similarly, the final benefit decreases when both benefit decrease or when either benefit decreases and the other stays. However, the final benefit is not decided when one increases while the other decreases.

3. The aggregation of final benefit is calculated based on the chosen disjunction operation and all outputs of rules. For example, the chosen operation is minimum t-norm $T_{min}(a, b, \ldots) = \{a, b, \ldots\}$. Then, the final benefit result function is $\mathrm{rf}(x) = T_{min}(\ldots, 0.4409 \times \mathrm{mf}_{medium}^{final}, \ldots)$. The final benefit value is the center of $\mathrm{rf}(x)$:
$$b = \frac{\int \mathrm{rf}(x) x \mathrm{d}x}{\int \mathrm{rf}(x) \mathrm{d}x}. \tag{4.39}$$

Similarly to benefit aggregation, fuzzy inference is also applied to aggregate offloading risks and the final risk is denoted as r. Both b and r are within range $[0, 1]$.

4.2.2 Decision Solution

The final benefit aggregated by user preference depends on mapping matrix X and the final risk. Thus b and r in previous section are actually $b(X)$ and $r(X)$. The offloading problem is to find mapping X to maximize the aggregate benefit, and meanwhile, the constraint is satisfied that the aggregate risk is smaller than the benefit:

$$\max b(X), \tag{4.40}$$

$$\text{s.t. } b(X) > r(X). \tag{4.41}$$

The offloading decision considers trade off of benefit and risk. The offloading is allowed only when the offloading motivation or benefit overwhelms the offloading risk.

To solve the above problem, we design an ant-algorithm [Taillard and Gambardella, 1997] based approach in Algorithm 3.

Algorithm 3 Algorithm: overview.

```
1: Initialization
2: repeat
3:     Solution construction
4:     Local search
5:     Pheromone update
6: until stopping criteria is reached
```

The previous problem can be represented by graph $G = (\Theta, \Lambda)$ where $\Theta = \{C, S\}$. Edges connect components to sites. An ant at component vertex chooses site vertex as next vertex to go and leaves pheromone on the trail.

The algorithm maintains a counter γ and a pheromone matrix τ. At the beginning, γ is set to 0, and τ_{ij} is set to 1 where $1 \le i \le m$ and $1 \le j \le n$. Then, three steps are executed in a loop until the result is reached. γ increases by 1 each round in loop. τ is used to guide the transition in solution construction step, and is updated in pheromone update step. Two stopping criteria are: the counter reaches γ_{max}, and the result stagnates. When either criterion is met, the algorithm stops.

Solution Construction

The solution construction is shown in Algorithm 4. The ant goes from component i to site j with the probability:

$$p_{ij}(\gamma) = \frac{\tau_{ij}(\gamma)}{\sum\limits_{1 \le k \le n} \tau_{ik}(\gamma)}. \tag{4.42}$$

A The ant algorithm is based on the following philosophy: high pheromone accumulated on the edge indicates this assignment is a potential good assignment; the ant

Algorithm 4 Algorithm: solution construction.

```
1: Set $X$ according to predefined assignment, and put unmovable component index
   into set $I$
2: while $|I| \leq m$ do
3:     Choose $i$ uniformly at random, where $1 \leq i \leq m$ and $i \notin I$
4:     Choose $j$ randomly with probability in (4.42), where $1 \leq j \leq n$
5:     $x_{ij} \leftarrow 1$
6:     $I \leftarrow I \cup \{i\}$
7: end while
```

chooses a potential good edge intentionally. However, the ant does not give up the choices of other edges.

Improvement Procedure

The neighborhood X' is obtained by changing one assignment in the solution X. The improvement procedure searches the neighborhood area and finds the valid local optimal solution as shown in Algorithm 5. The local search improves X through iterations. The parameter α controls the iteration times. The high α means the local search algorithm will try to explore more neighbors. The value range of α is $[1, m]$.

Pheromone Update

The elements of pheromone matrix are updated as:

$$\tau_{ij}(\gamma + 1) = \tau_{ij}(\gamma) + \Delta\tau_{ij}(\gamma), \tag{4.43}$$

Algorithm 5 Algorithm: local search.

```
1:  $I \leftarrow \emptyset$
2:  for  1 to $\alpha$  do
3:      Choose $i$ uniformly at random, where $1 \le i \le m$ and $i \notin I$
4:      $J \leftarrow \emptyset$
5:      while $|J| \le n$ do
6:          Choose $j$ uniformly at random, where $1 \le j \le n$ and $j \notin J$
7:          $X' \leftarrow X$, and change assignment from $i$ to $j$ in $X'$
8:          if $b(X) < b(X')$ and $b(X') > r(X')$ then
9:              $X \leftarrow X'$
10:         end if
11:         $J \leftarrow J \cup \{j\}$
12:     end while
13:     $I \leftarrow I \cup \{i\}$
14: end for
```

where

$$\Delta\tau_{ij}(\gamma) = \begin{cases} \eta\lambda_\gamma & \text{if } x_{ij} = 1 \text{ in current } X_\gamma \\ \eta^* & \text{if } x_{ij} = 1 \text{ in best } X_{best} \\ \eta\lambda_\gamma + \eta^* & \text{if } x_{ij} = 1 \text{ in } X_\gamma \text{ and } X_{best} \\ 0 & \text{otherwise} \end{cases} . \tag{4.44}$$

Two parameters η and η^* control the search scheduling. η is set to 1 at the beginning and varies in the run, while η^* is fixed in the whole process. In two cases, pheromone updates in another way that is different from (4.44):

1. When the best result ever found X_{best} has been improved by current solution X_γ, the η is reset to 1 and all τ_{ij} are reset to 1. The resetting remove historical information and intensify the search around new direction X_{best}.

2. When current solution X_γ reaches X_{best}, which means the focus on X_{best} is to too high, η is increased by 1 and all τ_{ij} are reset to new η to diversify the search.

The risk constraint in the problem is treated by parameter λ. When the solution X does not obey risk constraints, the positive feedback on pheromone is removed by paying penalty:

$$\lambda_\gamma = \begin{cases} 0 & \text{if } b(X_\gamma) < r(X_\gamma) \\ 1 & \text{otherwise} \end{cases} . \tag{4.45}$$

4.2.3 Implementations and Evaluations

This section discusses the implementation issues and presents the evaluation results. We evaluate our ant based algorithm by simulation in MATLAB. We first generate 50 test cases randomly, and then evaluate performance impact of algorithm parameter η^* and α. The η^* indicates the pheromone amount accumulated on links of best solution ever found, and α is the repeat times for neighbor exploring. These two parameters control the convergence speeds in global and local search, thus they are used to tune algorithm performance.

Risk estimation in practice

In the previous section 4.2.1 on offloading risks, there are several parameters that need estimation in practice. The parameter estimation can be done in many ways. As an example, these parameters can be estimated by subjective inference or training fed by time series of observation.

Trust estimation To estimate the probability of sites and links, subjective trust average can be used in practice. The trust rating of a site or a link is given by survey of clients that involved in the transaction with the site or link. The value of rating is an integer in the rating space $[0, k]$. For example, like what we usually used in rating mobile application in market, the rating space may be $[0, 5]$ stars and the rating of a site may be two stars. To normalize the rating to the range $[0, 1]$, the trust rating is scaled proportional to the ranting range. For example, if a site is rated as two stars and the rating range is five stars, the normalized rating is $2/5 = 0.4$. Assume the normalized ratings are $\{r_0, r_2, \ldots, r_k\}$ and the count of each rating is $(x_0, x_2, \ldots, x_k)$ in the sample from survey, then the sample size is $n = \sum_{i=0}^{k} x_i$ and the average rating is $\hat{r} = \sum_{i=0}^{k} r_i x_i / n$. Once the average trust is obtained, the function $p_1(v)$ in equation (4.33) is the complementary of the average trust $p_1(v) = 1 - \hat{r}(v)$.

Availability estimation The site and network availability can be estimated from time series observation. The network availability can be detected from operating system, while the site availability can be detected by heartbeat packet. Assume the availability of either site or network may be *on* or *off* state. The observation of states along with time is a series of *on*, *off*, *on*, *off* $\ldots$, which is an alternating renewal process [Wu *et al.*, 2013]. Figure 4.2 shows how availability changes along with time coordinate. Solid line represents *on* state duration, while dash line represents *off* state duration. Two states alternate with each other. One *on* duration and one *off* duration form a cycle. The *on* state duration is denoted as ξ and the *off* state duration is denoted as η. $\{\xi_i, i = 1, 2, \ldots\}$ is independent and identically distributed (i.i.d.), and so is $\{\eta_i, i = 1, 2, \ldots\}$. And ξ_i and η_j are independent for any $i \neq j$, but ξ_i and η_i can be dependent [Pham-Gia and Turkkan, 1999]. The cycle duration is denoted as χ and $\chi_i = \xi_i + \eta_i$ where $i = 1, 2, \ldots$. The proportion of *on* duration in any individual cycle is a random variable denoted as $\rho = \xi/\chi$. Then $p_2(v_i)$ in equation (4.36) is $p_2(v) = 1 - E(\rho)$.

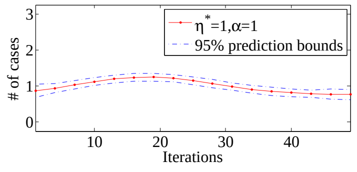
Evaluation of ant-based decision making algorithm

We generate 50 test cases for evaluation. In each case, parameters for time, energy, privacy and reliability factors are generated. For time factor, one application graph and one surrogate graph are generated. The application component number is set to 5 and the candidate surrogate site number is set to 3. The workload $\mathbf{w}$ and site processing ability $\mathbf{u}$ are generated uniformly at random in range $[1, 100]$ and $[1, 10]$ separately. The data exchange amount F and network throughput D are uniformly random distributed in range $[1, 10]$ and $[1, 100]$ separately. This parameters are set in order to meet the assumption that the application is computation intensive as the average time spent on computation is much greater than data exchange. For energy factor, the coefficients of mobile devices are similar to [Wang *et al.*, 2013]. The CPU coefficients of dynamic and static parts are random in range $[4, 6]$ and $[1, 1.5]$ separately. The RF coefficients of dynamic and static parts are random in range $[2, 3]$ and $[1, 1.5]$ separately. For privacy and reliability factors, the event probabilities are random in range $[0, 0.2]$. These events are assumed to happen in low probability in real situations.

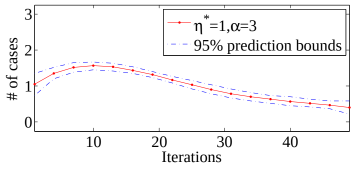
Algorithm tuning by η^* In the proposed algorithm, there are two parameters η^* and α for performance tuning. The parameter η^* indicates the search direction in the whole solution space. In every iteration of searching, the best solution ever found is always assigned more pheromone, which guides the ant to go to that direction in high probability in following iterations. The performance impact of η^* is shown in Figure 4.12. The figure shows the algorithm performance when α values are fixed at

1, 3 and 5. In the figure, the y axis represents the average case number and the x axis represents the iteration sequence number when the solution X is found. Every point (x, y) in the figure represents x cases reach their final solutions in the y^{th} iteration. The lines in the figure represent the case numbers distributed in iteration number range $[1, 50]$. Since the total case number is 50, the area below each line should be 50.

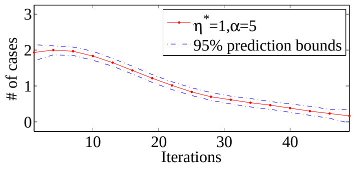
In all three situations of $\alpha = 1, 3, 5$, the line $\eta^* = 5$ is higher than $\eta^* = 3$ and the line $\eta^* = 3$ is higher than $\eta^* = 1$ in iteration range $[1, 20]$. And meanwhile, the line $\eta^* = 5$ is lower than $\eta^* = 3$ and the line $\eta^* = 3$ is lower than $\eta^* = 1$ in iteration range $[25, 50]$. When η^* raises, more cases that are solved originally in higher iteration numbers $[25, 50]$ will be solved in less iterations, which leads to the case number increases in range $[1, 20]$. When η^* increases, more pheromone will accumulate on the links that belong to the best solution X_{best}. This accumulated pheromone guides the ant to pick those links more often than other links, so that the ant goes to the direction of the X_{best}. The higher η^* is, the better sense of direction the ant has, which avoid ant's heading in random direction and save iterations. By comparing figures of the same α values, we also find that the more cases reach solutions in small iteration from line $\eta^* = 1$ to line $\eta^* = 3$ than from line $\eta^* = 3$ to line $\eta^* = 5$. This is obvious in range $[1, 10]$: the incremental case number between line $\eta^* = 1$ and line $\eta^* = 3$ is larger than that between line $\eta^* = 3$ and line $\eta^* = 5$. The incremental case number decreases when the η^* increases, which will finally lead to the limit situation when increasing η^* does not make iteration number smaller. When η^* is too high, the algorithm will always search around the point X_{best} in solution space and often hit it again. When algorithm hits the X_{best} again, the relative high value of η^* is decreased by the way of increasing η in case 2 of pheromone update step in algorithm to guarantee the search diversity.



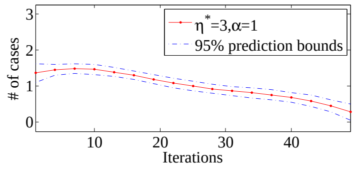
(a) $\eta^* = 1, \alpha = 1$



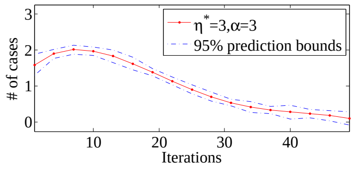
(b) $\eta^* = 1, \alpha = 3$



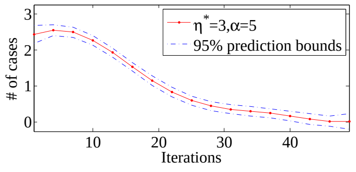
(c) $\eta^* = 1, \alpha = 5$



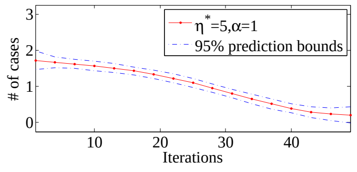
(d) $\eta^* = 3, \alpha = 1$



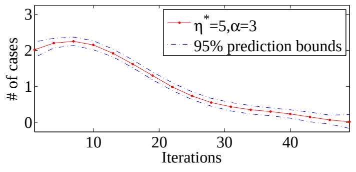
(e) $\eta^* = 3, \alpha = 3$



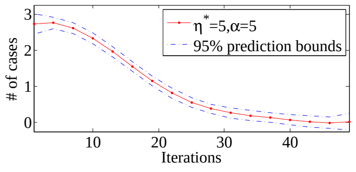
(f) $\eta^* = 3, \alpha = 5$



(g) $\eta^* = 5, \alpha = 1$



(h) $\eta^* = 5, \alpha = 3$



(i) $\eta^* = 5, \alpha = 5$

Figure 4.12: Performance Tuning.

Algorithm tuning by α While parameter η^* guides the global search, the parameter α controls the local search. The performance impact of α is also shown in Figure 4.12. The figure shows three situations: $\eta^* = 1$, $\eta^* = 3$ and $\eta^* = 5$. In all three situations, three α values are evaluated and compared. In Figure 4.12a,4.12b and 4.12c, the case number from line $\alpha = 1$ to $\alpha = 3$ and from line $\alpha = 3$ to $\alpha = 5$ increases in range $[1, 12]$ and decreases in range $[23, 50]$, which means some cases that are solved originally in high iteration numbers are solved in less iterations. This is obvious in iteration range $[1, 12]$ where the line $\alpha = 3$ is above $\alpha = 1$. Similar phenomenon is also seen in Figure 4.12d,4.12e,4.12f and Figure 4.12g,4.12h,4.12i. When α increases, the algorithm searches the near solution around the picked X. The near solution is the solution that has only one different assignment of component to site. Since the component number is limited, the largest α value is bounded by component number. The higher the α is, the more neighbors the algorithm explores in each iteration. The more neighbors the algorithm explores in each iteration, the earlier the algorithm finds the best solution in the dedicated area, which leads to improvement of performance from iteration number aspect. However, this improvement is bounded by two reasons. First, the α value is bounded to component number. Second, the local search costs more time when we increase local search quality. Although the iteration number is decreased, the time spent for each iteration is increased because high α requires more attempts to explore neighbors.

4.3 Offloading Series in Long Time Horizon

The previous sections and many existing approaches (e.g., [Giurgiu *et al.*, 2009] [Wolski *et al.*, 2008][Xian *et al.*, 2007] [Kemp *et al.*, 2012]) only focus on solving the one-time service composition topology configuration without considering a sequence of service composition due to the application running environment changes (e.g., due to the mobility of nodes). When considering multiple service composition decisions, one decision may impact other service mapping decisions at an up-coming service mapping. This demands that the topology reconfiguration decision must not only consider the current environment state but also predicate future environment changes, and thus, the service-topology reconfiguration issue can be modeled as a decision process issue. To address the above described issues, we model the service composition topology reconfiguration as a five-element tuple system and we present three algorithms to solve these decision process problems for three mobile cloud application scenarios: (1) finite horizon process: the application execution time is restricted by a certain time period, (2) infinite horizon process: there is no clear boundary of the application execution time, and (3) large-state situation: the large numbers of many-to-many service mappings demand a parallel computing framework to compute the service mapping decision.

The contributions of this research work are highlighted as follows:

- *Service composition topology reconfiguration process model.* We propose a theoretical model that describes the service composition topology reconfiguration process. The proposed model deals with a series of decisions rather than one-time decision. Particularly, the mobile device energy is used as the aggregated metric, considering network reliability, delay, CPU usage, etc., to model the service composition objective and reconfiguration reward.

- *Service composition topology reconfiguration algorithms.* Based on the proposed model, three algorithms are proposed to derive the service remapping decisions for three different service mapping scenarios: *finite horizon*, *infinite horizon*, and *large-state space* that represent ad hoc, long-term, and large-scale mobile cloud service composition scenarios, respectively.

4.3.1 System and Model

The offloading as a service and service composition in mobile cloud computing achieve three essential tasks. *First*, the mobile application components, which can be offloaded, are partitioned into several groups. *Second*, the surrogate sites are picked from candidate sites that are usually virtual machines in cloud. *Third*, the component groups in the first task are mapped to surrogate sites in the second task, and then computation is offloaded. The service composition is not a one-time process. Instead, the mapping in the third task should be dynamically changed to adapt the system and environment changes, especially in mobile cloud computing where mobility adds the environment variation.

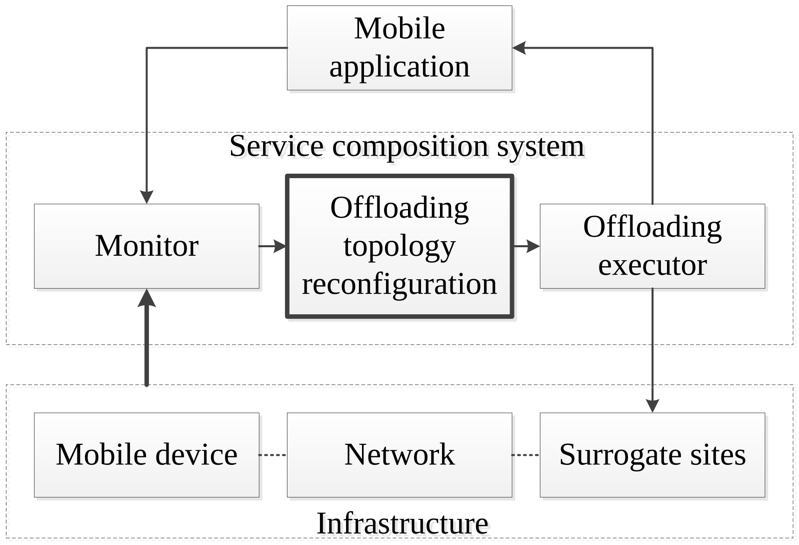


Figure 4.13: Continuous Service Composition System.

Service composition system

The *Service composition system* consists of three parts shown in Fig. 4.13, which can be emulated as human decision processes. The *Monitor* represents the human's eyes watching both the mobile applications and infrastructure, and notifying the *Service composition topology reconfiguration* their states's changes. The *Service composition topology reconfiguration* is the human's brain thinking about how to put application components onto which surrogate sites. The *Service composition executor* represents the human's hands that enforce the component-surrogate mapping and make sure a mobile device and its surrogates work together properly. These three interdependent modules can be deployed in a cloud computation platform to provide offloading-as-a-service, which exposes http REST API for mobile applications and the infrastructure to interact with. The infrastructure registers itself to the service and push the state information to the *Monitor* The mobile application as well registers itself to the service and receives the execution command from the *Service composition executor.*

Besides the *Service composition system*, the *Mobile application* and *Infrastructure* are involved in the service composition process, and they form a loop where the *Mobile application* and the *Infrastructure* share a part of the loop in parallel. The *Service composition system* initiates the service composition operations that stimulates the *Mobile application* and the *Infrastructure*. The *Mobile application* and the *Infrastructure* integrate the service composition impacts and the variances form the outside, such as user inputs to the application and the device load variance, and feedback to the *Service composition system*. Then, the *Service composition system* based on the feedback makes the service composition operation.

The *Mobile application* is the objective of the *Service composition system*. The *Infrastructure* consists of the three players that involved in the service composition process: the *Mobile device*, the *Surrogate sites* and the *Network* between them. The following sub subsections model the *Mobile application*, the *Surrogate sites* and the service composition mapping between them.

Model statement

Several factors involved in the service composition topology reconfiguration are modeled as follow.

Decision points Decision points are the moments when the service composition topology reconfiguration decisions are made. At these moments in Fig. 4.13, the *Monitor* triggers the *Service composition topology reconfiguration* for generating a service composition topology. The moments are decided by the *Monitor* that is always observing the *Infrastructure* and the *Mobile application* states. The *Monitor* can triggers decisions periodically according to a predefined period Δt.

The decision points are a sequence that starts from the time when the application starts to the time when the application ends: $T = \{0, 1, 2, \dots, N-1\}$. In some scenarios, the application termination time is not expectable, the sequence is an infinite sequence: $T = \{0, 1, 2, \dots\}$.

Measures The *Monitor* measures the *Infrastructure* states. The node computation capability and link throughput are labeled on the surrogate network, which transformed G_{sur} into a weighted graph. The available computation capability on a candidate site s is labeled as a node weight u_s and vector $\mathbf{u}$ is weights of all sites. The network throughput on link $l = (s_1, s_2)$ from s_1 to s_2 is labeled as link weight $d_{s_1 s_2}$ and matrix $D_{n\times n}$ is weights of all links. The diagonal of D are large values because the delay in the same site can be considered negligible.

The above measures may be continuous variables. They are manipulated by normalization and quantization to make them into discrete states. A relative large value is picked and the observation is scaled to be in $[0, 1]$. Quantization precision decides the size of state space. The state space is denoted as S.

Service composition topology The service composition topology is the effective component surrogate mapping $X_{\tilde{m}\times n}$. Each mapping is corresponding to an action that enforces the mapping. The size of service composition topology space is $n^{\tilde{m}}$ since each component has n choices and there are $\tilde{m}$ components that choose their choices independently. Let A be the corresponding action space from which the *Service composition system* in Fig. 4.13 acts - picking and enforcing the mappings. The output of the *Service composition topology reconfiguration* is a service composition topology. If the outputted service composition topology is different from the old one, reconfiguration is needed. The 'reconfiguration' means the components in the old service composition topology should be moved properly to satisfy the new service composition topology, which is fulfilled by the *Service composition executor*.

Observation learning The *Service composition topology reconfiguration* in Fig. 4.13 counts the historical states and actions, and estimates the state transition probability $p(j \mid i, a)$ where $i, j \in S$ and $a \in A$. The transition probability satisfies $\sum_{j \in S} p(j \mid i, a) = 1$. This probability is updated at decision points. The *Service composition topology reconfiguration* maintains a buffer that keeps the count for valid observed states and the count of historical actions. At each decision point, it gets the current state j from the *Monitor* and, the last state i and the last topology decision a from its buffer. Then it calculates the state transition probability for every pair of states with every action. This transition probability that comes from the recent history is used to predict the probability of transition for the near future Δt period assuming the transition probability stays steady in short time period.

Service composition objective One of the major service composition goals in mobile cloud computing is to lengthen the battery life on mobile devices. We use energy consumption as the offloading objective in the equation 4.32 in this decision scenario.

When a topology is picked at a decision point, a reward value is calculated to indicate how good the decision is. The reward of choosing action a at state i depends not only state i and action a but also the next state j. The reward is:

$$r(i, a) = \sum_{j \in S} r(i, a, j) p(j \mid i, a), \tag{4.46}$$

where the reward of transition from state i to state j with action a is the expected delta power between two states i and j with the same action a, and

$$r(i, a, j) = P(j, a) - P(i, a). \tag{4.47}$$

Model formulation The service composition topology reconfiguration process is modeled as a five element tuple:

$$\{T, S, A, p(\cdot \mid i, a), r(i, a)\}. \tag{4.48}$$

This is a Markov decision process [Liu, 2004].

4.3.2 Design and Algorithm

Based on the proposed service composition topology reconfiguration system and model, this section formulates the topology reconfiguration policy problem in both finite horizon scenario and infinite horizon scenario. The solutions in both scenarios are presented. Besides, the MapReduce based algorithms are discussed for large state count situations that are common in real world.

Topology reconfiguration policy

The service composition reconfiguration policy is a function π that maps states to actions $\pi : S \to A$. The function π can be stored in memory as an array whose index is the state and whose content of each element is the corresponding action. Let Y_t and Δ_t be the system state and the picked action at decision point t: $\Delta_t = \pi(Y_t)$. The sequence $L(\pi) = \{Y_0, \Delta_0, Y_1, \Delta_1, \dots\}$ is a stochastic process depending on π. Let $R_t(\pi) = r(Y_t, \Delta_t)$, then the sequence $\{R_0(\pi), R_1(\pi), \dots\}$ is a reward process depending on π.

Finite decision points At any decision point, the current period reward could be used as decision goal, which, however, is shortsighted because the maximum current period reward does not guarantee that the sum of rewards in all periods is the maximum. To make proper decision on service composition topology, the total rewards of N periods should be considered as the goal:

$$V_N(Y_0, \pi) = \sum_{t=0}^{N-1} \beta^t R_t(\pi) = \sum_{t=0}^{N-1} \beta^t r(Y_t, \Delta_t), \tag{4.49}$$

where β is the confidence index. As the N periods' rewards are estimated future rewards, the degree of confidence on the reward sequence $R_t(\pi)$ is decreasing along with t goes far from now. The confidence index presents this decreasing confidence trend. The service composition topology reconfiguration problem is to find the policy π that maximizes $V_N(Y_0, \pi)$.

Let u_t be the reward sum from decision point t to N. There is backward recursive relation between $t+1$ and t:

$$u_t(i_t, a_t) = r(i_t, a_t) + \beta \sum_{j \in S} p(j \mid i_t, a_t) u_{t+1}(j) \tag{4.50}$$

$$= \sum_{j \in S} p(j \mid i_t, a_t)(r(i_t, a_t, j) + \beta u_{t+1}(j)), \tag{4.51}$$

where $i_t \in S$ is the state at time t. The equation (4.50) shows the reward sum from time t to N is consist of the current period reward and the β scaled reward sum from time $t+1$ to N.

Let the superscript notation * present the maximum value of the corresponding variable. To get the maximum rewards, the backward recursive relation formulation is:

$$u_t^*(i_t) = \max_{a_t \in A} \left\{ r(i_t, a_t) + \beta \sum_{j \in S} p(j \mid i_t, a_t) u_{t+1}^*(j) \right\}. \tag{4.52}$$

The Algorithm 6 shows the algorithm to calculate the policy π. The main body of the algorithm repeats the equation (4.52) by N times. In the algorithm, the line 5 and the line 6 could share the intermediate result of equation (4.50), which

Algorithm 6 Finite horizon backward induction.

```
1: $t \leftarrow N$
2: $u_t^*(i_t) \leftarrow 0, \forall i_t \in S$
3: while $t > 0$ do
4:     $t \leftarrow t - 1$
5:     $u_t^*(i_t) \leftarrow \max_{a \in A} \left\{ r(i_t, a) + \beta \sum_{j \in S} p(j \mid i_t, a) u_{t+1}^*(j) \right\}, \forall i_t \in S$
6:     $\pi_t^*(i_t) \leftarrow \arg\max_{a \in A} \left\{ r(i_t, a) + \beta \sum_{j \in S} p(j \mid i_t, a) u_{t+1}^*(j) \right\}, \forall i_t \in S$
7: end while
```

means $r(i_t, a) + \sum_{j \in S} p(j \mid i_t, a) u_{t+1}^*(j), \forall i_t \in S$ is calculated only once but used in both operations.

The algorithm requires storage for two arrays indexed by state: $\mathbf{v}$ and $\mathbf{f}$. The (i) element of the array $\mathbf{v}$ is $u^*(i)$ where $i \in S$. The (i) element of the array $\mathbf{f}$ is an action $\pi^*(i) \in A$ that is the optimal action corresponding to state $i \in S$. The array $\mathbf{f}$ hosts one instance of π. The length of both $\mathbf{v}$ and $\mathbf{f}$ is $|S|$.

At the end of the algorithm, $\mathbf{v}$ contains the discounted sum of the rewards to be earned on average from the corresponding initial state i: $V_N^*(i) = \mathbf{v}(i) = u_0^*(i)$ where $i \in S$. The policy is $\pi = \{\pi_0^*, \pi_1^*, \ldots, \pi_{N-1}^*\}$ for N decision points. The array $\mathbf{f}$ contains π_0^* when the algorithm ends. The action $\pi_0^*(Y_0) = \mathbf{f}(Y_0)$ is the action that should be performed at the current decision point.

Infinite decision points The previous discussion of the finite horizon scenario can be extended to the infinite horizon scenario. The objective function for the infinite horizon scenario can be achieved by pushing N to ∞ in the equation (4.49):

$$V(Y_0, \pi) = \lim_{N \to \infty} V_N(Y_0, \pi). \tag{4.53}$$

Similarly, the problem is to find the policy π that maximizes the total rewards $V(Y_0, \pi)$. In the infinite horizon scenario, the recursive relation (4.50) is generalized by removing iteration subscript:

$$u(i,a) = r(i,a) + \beta \sum_{j \in S} p(j \mid i,a) u(j). \tag{4.54}$$

Similarly, the recursive relation (4.52) changes to:

$$u^*(i) = \max_{a \in A} \left\{ r(i,a) + \beta \sum_{j \in S} p(j \mid i,a) u^*(j) \right\}, \tag{4.55}$$

which means the $u^*(i)$ converges to the maximum total rewards.

The Algorithm 7 shows the algorithm to calculate the policy π. Compared to Algorithm 6, the iteration termination condition is changed to comparing the vector norm and tolerance. The ε indicates the tolerance for the converging state. In the line 6, the $\|\cdot\|$ is vector norm that could be any type of L_p: L_1, L_2, or L_∞ norm. In addition, a local improvement loop is added inside the main iteration. The sequence $\{m\}$ is consist of non-negative integers that are used in each iteration as improvement depth. The $\{m\}$ could be generated in many ways. For example, it may be constant: $m_n = m$, or it may get more precise along with the iteration sequence number: $m_n = n$. When the algorithm ends, the policy is π_{n+1} that is stored in the array **f**.

Two column vector $\mathbf{r}(\pi)$ and a matrix $P(\pi)$ are defined to simplify expressions in the algorithm. The (i) element of vector $\mathbf{r}(\pi)$ is $r(i, \pi(i))$ where $i \in S$. The size of $\mathbf{r}(\pi)$ is $|S|$. The (i,j) element of matrix $P(\pi)$ is $p(j \mid i, \pi(i))$ where $i, j \in S$. The size of $P(\pi)$ is $|S| \times |S|$. In the algorithm, the line 13 $\sim$ 15 repeat the same operations as the line 3 $\sim$ 5. The line 8 is the vector version of equation (4.54). The line 5 and 15 are the vector version of equation (4.55). The line 3 and the line 5 share the intermediate computation result. Similarly, the line 13 and the line 15 also share the intermediate computation result.

Algorithm 7 Infinite horizon induction.

```
1: $n \leftarrow 0$
2: $\mathbf{v}^n \leftarrow \mathbf{0}$
3: $\pi_{n+1} \leftarrow \arg\max_{\pi \in \Pi}\{\mathbf{r}(\pi) + \beta P(\pi)\mathbf{v}^n\}$
4: $k \leftarrow 0$
5: $\mathbf{u}_n^k \leftarrow \max_{\pi \in \Pi}\{\mathbf{r}(\pi) + \beta P(\pi)\mathbf{v}^n\}$          ▷ Equation (4.55)
6: while $\|\mathbf{u}_n^k - \mathbf{v}^n\| < \varepsilon$ do
7:     while $k < m_n$ do          ▷ Equation (4.54)
8:         $\mathbf{u}_n^{k+1} \leftarrow \mathbf{r}(\pi_{n+1}) + \beta P(\pi_{n+1})\mathbf{u}_n^k$
9:         $k \leftarrow k + 1$
10:     end while
11:     $\mathbf{v}^{n+1} \leftarrow \mathbf{u}_n^{m_n}$
12:     $n \leftarrow n + 1$
13:     $\pi_{n+1} \leftarrow \arg\max_{\pi \in \Pi}\{\mathbf{r}(\pi) + \beta P(\pi)\mathbf{v}^n\}$
14:     $k \leftarrow 0$
15:     $\mathbf{u}_n^k \leftarrow \max_{\pi \in \Pi}\{\mathbf{r}(\pi) + \beta P(\pi)\mathbf{v}^n\}$          ▷ Equation (4.55)
16: end while
```

Large state space

To make more accurate and agile decision, the real world measures are usually lead to large state space in section 4.3.1. The large state space size results in long responding time. To mitigate the responding time in large state space situation, MapReduce could be used. This section discusses the conversion from the Algorithm 6 and the Algorithm 7 to MapReduce algorithms.

The Algorithm 8 and Algorithm 9 show the MapReduce algorithms for the finite horizon scenario. The input to mapper function, which is also the output of reducer

Algorithm 8 Finite horizon Mapper.

```
1: function MAP($i$, $Q$)
2:     Emit ($i$, $Q$)                                  ▷ Pass state $i$
3:     for all $j \in S$ do
4:         for all $a \in A$ do
5:             Emit ($j$, $< i, Q, a >$)
6:         end for
7:     end for
8: end function
```

function, is the state id i and an object Q that encapsulates the state information. Besides encoded state information, two components $Q.v$ and $Q.f$ corresponding to the arrays **v** and **f** are also included in the state object. Moreover, a component $Q.p$ corresponding to the state transition probability is included in the state object as well. The state object is passed from the mapper to the reducer for calculating the equation 4.51. This is accomplished by emitting the state data structure itself, with the state id as a key in the line 2 in the mapper. In the reducer line 3, the node data structure is distinguished from other values.

The mapper function associates the current state with all backward states. The reducer function aggregates the reward sum of all forward states according to the equation (4.51), which is categorized by action. Then it picks the maximum reward sum and the corresponding action as the current reward sum and action according to the equation (4.52).

It is apparent that the algorithm in Algorithm 6 is an iterative algorithm, where each iteration corresponds to a MapReduce job. The actual checking of the termination condition must occur outside of MapReduce. Typically, execution of an iterative MapReduce algorithm requires a non-MapReduce “driver” program, which submits

Algorithm 9 Finite horizon Reducer.

```
function REDUCE($i$, $[< j, Q, a >]$)
    for all $x \in [< j, Q, a >]$ do
        if Is($x$) then                                    ▷ Recover state $i$
            $Q \leftarrow x$
        end if
    end for
    $set \leftarrow$ new HashSet
    for all $x \in [< j, Q, a >]$ do
        if IsNot($x$) then                                 ▷ Equation (4.51)
            $set(a) \leftarrow set(a) + Q.p(j, a)(r(Q, a, x.Q) + \beta x.Q.v)$
        end if
    end for
    for all $x \in set$ do
        if $Q.v < set(a)$ then                             ▷ Equation (4.52)
            $Q.v \leftarrow set(a)$
            $Q.f \leftarrow a$
        end if
    end for
    Emit ($i$, $Q$)
end function
```

a MapReduce job to iterate the algorithm, checks to see if a termination condition has been met, and if not, repeats [Lin and Dyer, 2010].

As presented in the section 4.3.2, the infinite horizon algorithm is obtained by extending the finite algorithm, the MapReduce based algorithm for the infinite horizon scenario can be obtained by extending the MapReduce based finite horizon algorithms. The mapper function could be used without modification in the infinite horizon algorithms. The improvement loop in the infinite horizon algorithm can be achieved by repeating the line $8 \sim 12$ $m + n$ times. Besides the modification on the reduce function, the modification on driver is required. The iteration termination condition in the driver is changed from fix number to the comparison of norm and coverage tolerance.

4.3.3 Evaluation

This section discusses the proposed models and presents the evaluations of the proposed algorithms including finite horizon, infinite horizon and MapReduce based algorithms.

Evaluation cases and default parameter setting

We generate 200 test cases of the proposed model in section 4.3.1. To make the mobile device profile based on solid ground, we pick up the mobile device profiling parameters obtained in previous work [Zhang *et al.*, 2010a]. The maximum dynamic power consumption values in each mobile CPU and RF components, c^{CPU} and c^{radio} , are normally distributed with mean of 4.34 and 710, and variation of 1.46 and 48, respectively. The static power consumption values in each mobile CPU and RF components, P^{CPU}_{idle} and P^{radio}_{base}, are normally distributed with mean of 121.46 and 20, and variation of 9.20 and 4.86, respectively. In our experiments, the CPU utility

Table 4.3: Default Parameter Setting for Offloading Topology Reconfiguration Evaluation.

Parameter		Default value
application and cloud	$\tilde{m}$	8
	n	5
	$\|S\|$	100
finite horizon	N	3
	β	0.7
infinite horizon	L_p	L_2
	m_n	n
	ε	1
large state space	$\|S\|$	1000

is uniformly distributed from 40% to 90%. The network throughput is uniformly distributed from 10% to 40% after normalization.

Besides the mobile device and cloud profiling, we use the default algorithm parameter values shown in Table 4.3. These values may be changed in the experiments to show their impact on the algorithms. We used MATLAB to evaluate the finite scenario and the infinite horizon scenario algorithms. The application graphs were randomly generated with the above parameters in MATLAB. The algorithms were implemented according to the Algorithm 6 and Algorithm 7 in MATLAB. For large state scenario algorithm evaluation, we used a Hadoop cluster. The algorithm was implemented according to the Algorithm 8 and Algorithm 9 in Java.

Experiments on finite horizon

We first evaluate the norm of the reward sum trend along with the variance of forecast decision point number N. Since the reward sum range of test cases vary, we normalize

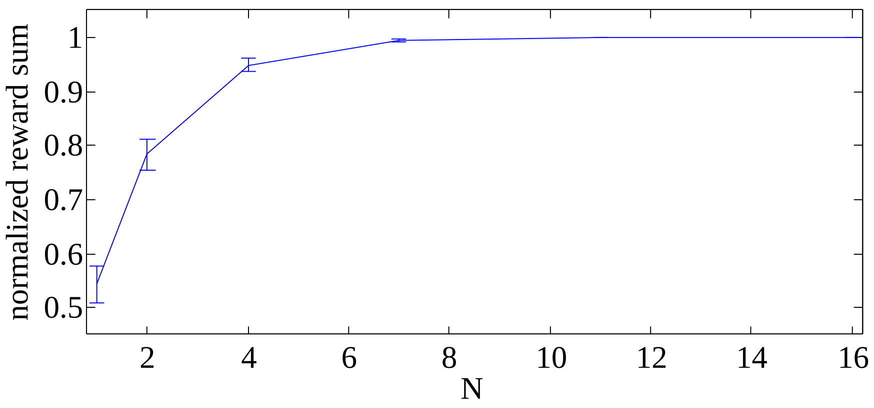


Figure 4.14: Reward Sum vs. N in the Finite Horizon Scenario.

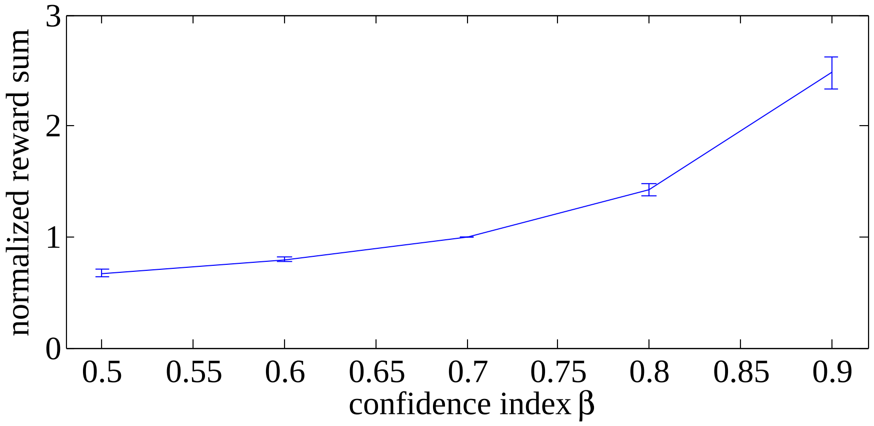


Figure 4.15: Reward Sum vs. β in the Finite Horizon Scenario.

the reward sum against the maximum reward sum in the evaluation. The experiment result is shown in Fig. 4.14. The trend in the figure shows the reward sum approaches the maximum reward sum along with the N increases. The approaching trend follows a monotonically increasing trend. When the reward sum approximate the stable value (N is greater than 8), we can claim the N is large enough to be considered as infinite horizon scenario.

We then evaluate the norm of the reward sum trend along with the variance of confidence index β. The experiment result is shown in Fig. 4.15. In the figure, the reward sum is normalized against the values of default confidence index. The reward sum increases monotonically along with the confidence index increase. The

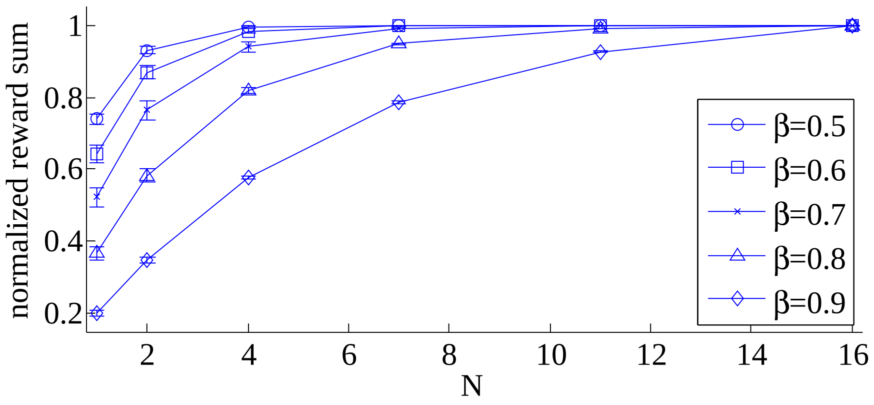


Figure 4.16: N and β Correlation in the Finite Horizon Scenario.

higher the confidence index increases, the more reward values are added to the total considered reward, which leads to the higher norm value of reward norm. The reward sum increases sharply when the confidence index approximates to 1, which leads to linear proportional relation between reward sum and the forecast period N or infinite horizon reward sum calculation failure. When the confidence index is small, the reward sum is steady because the reward sum is mainly consists of rewards in several near future forecast periods.

To illustrate the correlation of N and β, we put the reward trends of different confidence indexes in Fig. 4.16. In the figure, when the confidence index is small ($\beta = 0.5$), the reward sum converges after a short period ($N = 4$). When $\beta = 0.7$, the value for reward sum to converge ($N = 8$) is as twice as for $\beta = 0.5$. The higher the confidence index is, the larger the decision point number is to make the expected reward sum to converge. Since the N values for converge is related to the iteration times in the infinite scenario, we can infer that the smaller the confidence index is the less the iteration is required in the infinite horizon scenario.

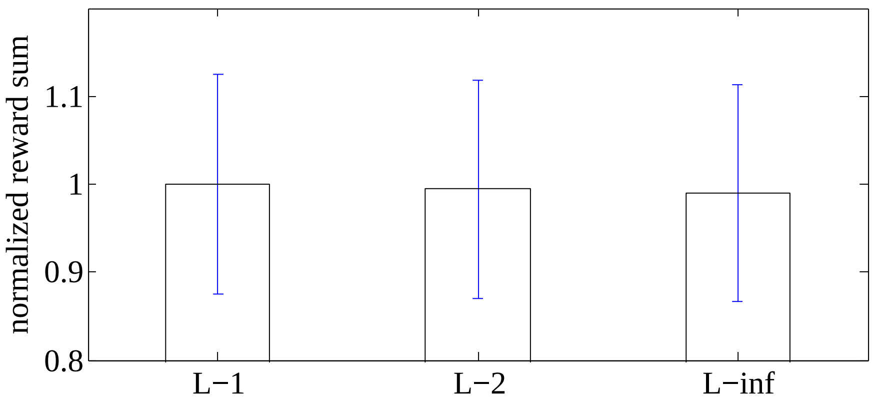


Figure 4.17: Reward Sum of Different Norms.

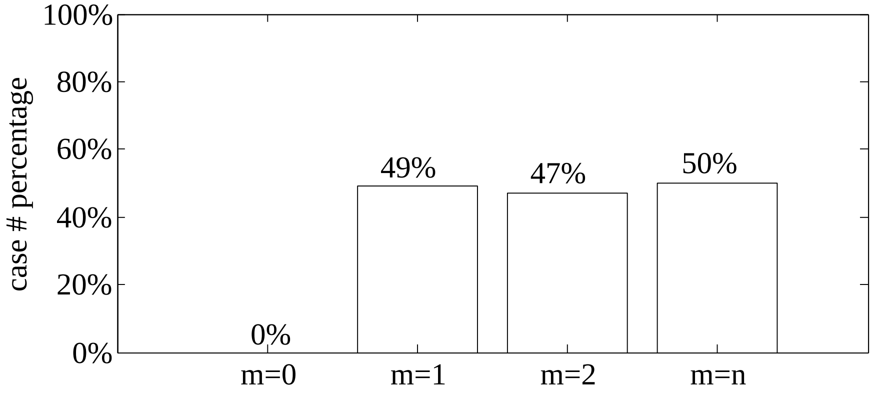


Figure 4.18: Iteration Cycles of Different $\{m_n\}$ Sequences.

Experiments on infinite horizon

In the infinite horizon algorithm, several types of norm could be used. We experiment L_1, L_2 and L_∞ for the algorithm and show the results in Fig. 4.17. In the figure, the y-axis is the reward sum normalized against the maximum reward sum in the experiment. The figure shows the results of three types of norms are almost the same. The difference is about 2% of the all reward sum, which could be ignored. The result demonstrates the algorithm does not depend on the norm types.

We evaluate the impact of different $\{m_n\}$ sequences on the iteration cycles in the infinite horizon algorithm and show the results in Fig. 4.18. Three $\{m_n\}$ sequences are used in the experiments: $m_n = 1$, $m_n = 2$ and $m_n = n$. We find that almost half cases benefit from the improvement procedure of line 8 in Algorithm 7 where the control

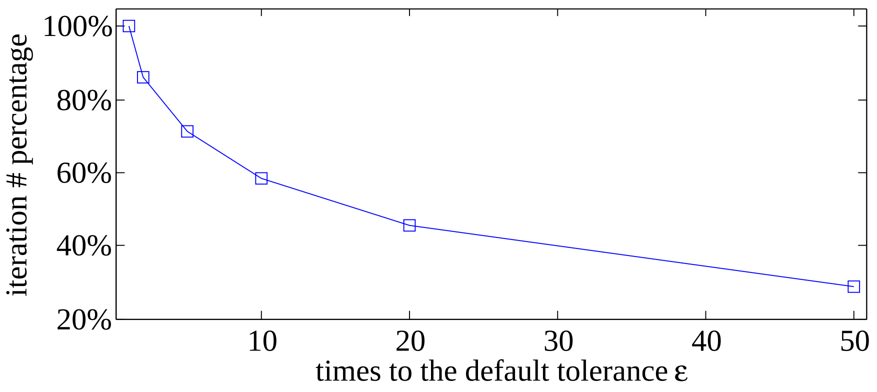


Figure 4.19: Reward sum vs. N in the finite horizon scenario.

case $m = 0$ is without improvement. The inside improvement loop uses the current states' information to estimate the reward sum before all states' information are updated by the outside main loop. Although the estimation based by current states' information may not be accurate, the trend is more or less captured by this estimation so that the outside main loop iteration cycles can be saved. The total computation performance benefit produced by the improvement procedure varies depending on the tradeoff of main loop saving and improvement loop cost. In the figure, we also find that the three strategies' performance are almost the same, which means the benefit of improvement loop does not depend much on the constants loop cycles. From the Fig. 4.14, we can see that the reward sum converge quickly along with the iteration cycles, so the reward sum after improvement loop changes dramatically compared to small tolerance value no matter how many loop cycles there are.

The main loop of infinite horizon algorithm terminates in condition controlled by the reward sum tolerance. We experiment different tolerance and their corresponding main loop iteration cycles and show the results in Fig. 4.19. In the figure, the x-axis is the ε range from 1 times ε to 50 times ε, and the y-axis is the iteration cycle number that is normalized against the cycle number of default ε situation. Obviously, the higher the tolerance is, the quicker the iteration ends. The large tolerance means

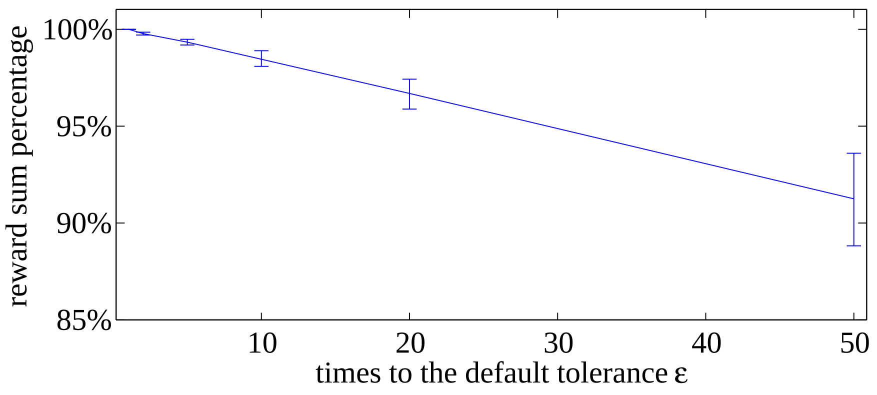


Figure 4.20: Reward sum of different ε.

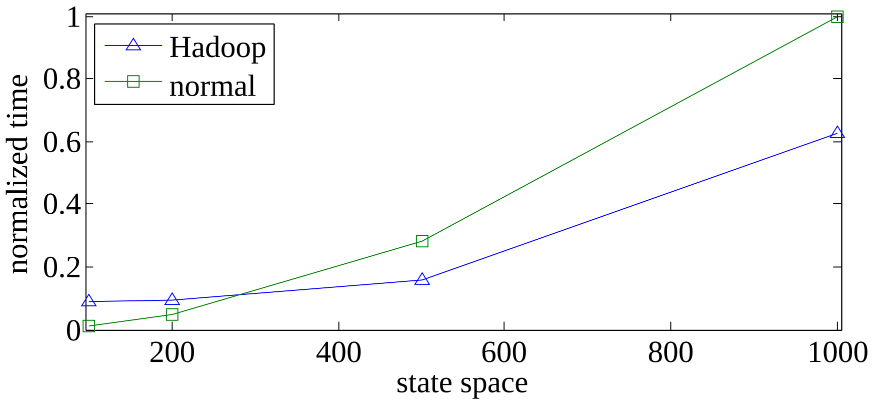


Figure 4.21: Execution time on Hadoop.

the more reward sum values fall into the area that is considered converged values, so the less iteration may let the reward sum trend go into that area and terminate the iteration cycle. We also illustrate the reward sum of different ε in Fig. 4.20. In the figure, similar to Fig. 4.19, the x-axis is the ε range from 1 times ε to 50 times ε, and the y-axis is the reward sum that is normalized against the cycle number of default ε situation. We can see that the reward sum decreases when the tolerance increases, which demonstrates that the coverage area is enlarged by large tolerance. Additionally, the reward sum decreases proportional to the tolerance variance, which is expected because the converge area is enlarged by turbulence proportionally.

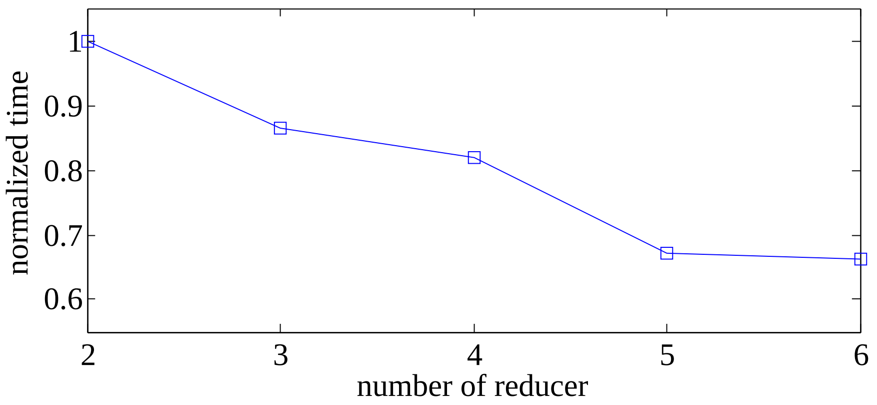


Figure 4.22: Execution time on different Hadoop configuration.

Experiments on large state space

We evaluate the MapReduce algorithm on Hadoop platform. The default Hadoop cluster configuration is 1 master node and 6 slave nodes. We experiment four state spaces of different sizes: 100, 200, 500 and 1000 for the finite horizon scenario. The result is shown in Fig. 4.21. The infinite horizon scenario experiment shows the similar results as Fig. 4.21. In the figure, the x-axis is the space size and the y-axis is the execution time normalized to the maximum time in the experiment. We can see in the figure that the execution time of both normal and Hadoop algorithm increase dramatically along with state space size increases. However, the execution time increase speed is slow than that of normal execution. Besides, two execution time lines intersect at around a point corresponding to the state space size of 300. Since the Hadoop spends some time on job management and coordination between servers, the execution of MapReduce version of algorithm does not have advantage when the state space size is small. The exact boundary point when the MapReduce execution starts to have advantage depends not only on the state space size but also the Hadoop configuration and running environment, so we should be careful to pick the proper execution method on real deployment.

The MapReduce algorithm performance is related to the reducer number. We have conducted experiments of the algorithm under Hadoop configurations with different reducer numbers. The normalized execution time is shown in Fig. 4.22. When the reducer's number increases, the execution time decreases because more parallel computation are applied.

Evaluation summary and discussions

Compared the evaluation results of the finite horizon to the infinite horizon scenarios, we summarize the findings as follows, which direct our future work: (1) The reward sum goes steady when the horizon window N goes infinite, which shows the process of quantitative change to qualitative change. (2) The larger confidence indicator β leads to longer converge window according to Fig. 4.16. (3) For the infinite horizon scenario, the proposed algorithm is flexible so that distance norms do not have a major impact on the system performance. (4) The improvement local search enhances the algorithm performance. (5) The converge tolerance ε controls the algorithm iteration cycle number, which is carefully chosen to balance the computation load and the calculation accuracy. Finally, (6) for the large state scenario, the MapReduce algorithm can achieve significant performance improvement beyond the normal algorithm.

In the presented evaluation, we tried to cover various aspects for mobile cloud service composition to provide an optimal decision making. However, there are some improvements could be done in the future work to enhance the proposed solution. In the current implementation, we assign a periodical time interval to run the decision making algorithms. The decision time point strategy can be more adaptive and intelligent, such as adaptive decision points according to the system status. Thus, a feedback model is needed to incorporate situation awareness into the considerations. Moreover, the service composition objective should include network reliability and risk into the consideration. Furthermore, the transition among the presented three algorithms in the mobile cloud service composition decision model needs further study.

Chapter 5

MIDAS: VEHICULAR CLOUD

This project demonstrates the synergistic use of a cyber-physical infrastructure consisting of smart-phone devices; cloud computing, wireless communication, and intelligent transportation systems to manage vehicles in the complex urban network through the use of traffic controls, route advisories and road pricing to jointly optimize drivers mobility and the sustainability goals of reducing energy usage and improving air quality. The system developed, MIDAS-CPS, proactively Manages the Interacting traffic Demand And the available transportation Supply. A key element of MIDAS-CPS is the data collection and display device PICT that collects each participating drivers vehicle Position, forward Images from the vehicles dashboard, and Communication Time stamps, and then displays visualizations of predicted queues ahead, relevant road prices, and route advisories.

Given the increasing congestion in most of the urban areas, and the rising costs of constructing traffic control facilities and implementing highway hardware, MIDAS-CPS could revolutionize the way traffic is managed on the urban network since all computing is done via clouds and the drivers instantly get in-vehicle advisories with graphical visualizations of predicted conditions. It is anticipated this would lead to improved road safety and lesser drive stress, besides the designed benefits on the environment, energy consumption, congestion mitigation, and driver mobility. This multidisciplinary project is at the cutting edge in several areas: real-time image processing, real-time traffic prediction and supply/demand management, and cloud computing.

The MIDAS project consists of three research groups: mobile cloud group, image processing group, and intelligent traffic group. The mobile cloud group is responsible for the mobile cloud communication, resources scheduling, and applications.

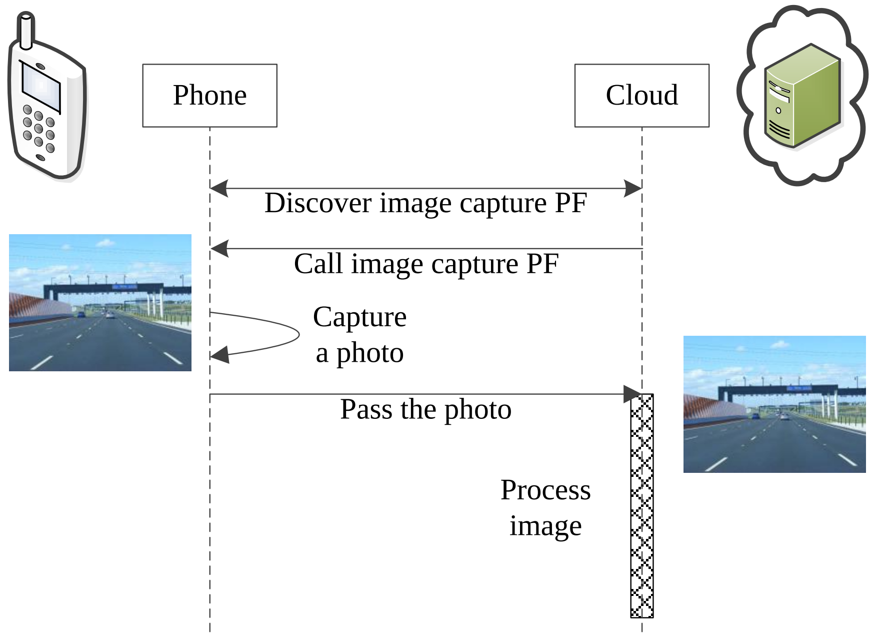
5.1 Remote Image Capture Based on POEM

The POEM framework is used in the MIDAS project. The motivation example in Figure 1.1 is one of the scenarios that MIDAS system works. This section describes POEM evaluation using the motivation scenario application in Figure 1.1.

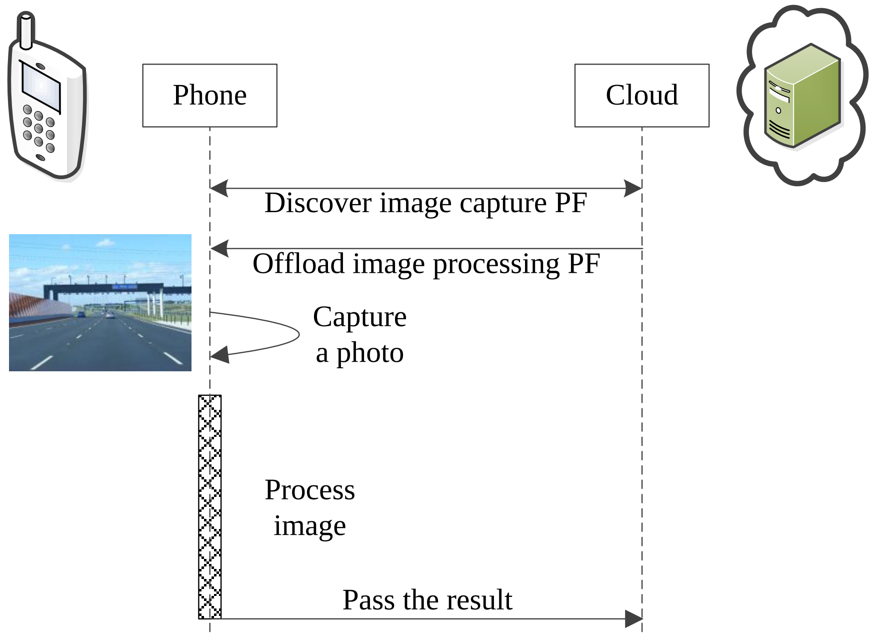
5.1.1 Methodology

In the motivation example, Both Alice and Bob provide the image capture service, and Bob and Carol consume the image from the remote service. There are two types of interactions in the motivation example:

- Interaction between phone and VM: Bob would like to know the traffic density of the road ahead. The vehicle density is a float number indicating how many vehicles in a road segment. A simple way to calculate the vehicle density in the motivation example is first filtering the objects in the image and then counting the vehicles simply. As summarized in the section 3.1.1, we can implement this type of interaction either by service composition in Figure 5.1a, which transfers the image data from Alice to Bob while keeps the PF location at the Bob vehicle, or by computation offloading in Figure 5.1b, which migrates the image process bundle from Bob to Alice, runs the image processing at the Alice vehicle and transfers only small amount of output data from Alice to Bob.
- Interaction between phone and phone: Carol would like to know the traffic density as well. Both Alice and Bob can provide the image capture service. Similarly, Carol may request the image directly sent from phone to the him



(a) Service composition.



(b) Computation offloading.

Figure 5.1: Interaction Between Phone and VM.

and he can calculate the vehicle density, as service composition; or the phones on the vehicles may calculate the vehicle density that is sent to Carol, as service offloading. The difference between this type interaction and Figure 5.1 is replacing the VM to phone.

We use Android phone to represent Alice and Bob, and use a Ubuntu VM to represent Carol. Thus our evaluation includes four cases: service composition between phone and VM, computation offloading between phone and VM, service composition between phone and phone, and computation offloading between phone and phone.

The POEM Manager is implemented on Felix [Gédéon, 2010] OSGi implementation version 4.0.3. Mobile application that contains a Felix OSGi framework instance that hosts POEM Manager runs on Android Motorola phone A855. The phone's parameters are 600MHz CPU and 256M memory. The Android version is 2.2.3. The virtual machine is with 1GHZ CPU and 512M memory, which runs Ubuntu 11.10.

The phone and the cloud server are connected by a router that is also wifi access point for the phone. The WiFi connection has averaged latency of 70 ms, download bandwidth of 7 Mbps, and upload bandwidth of 0.9 Mbps. Ping is used to report the average latency from the phone to the ESSI, and Xtremelabs Speedtest, downloaded from Android market, is used to measure download and upload bandwidth.

The image process part is implemented based on OpenCV [Bradski and Kaehler, 2008] library. We used the bilateral filter as the example PF for image processing. Bilateral filter can reduce unwanted noise very well while keeping edges fairly sharp. One of the parameters to the bilateral filter is the filter size that is the diameter of each pixel neighborhood that is used during filtering. We use 10 as the default filter size. We run the application against different filter size. The test image size is around 45 KB and it is with 800×600 pixels.

Experiment result is obtained by running the application 50 times for every scenario and averaged. Between two consecutive executions there is a pause of 1 second.

5.1.2 Interaction Between Phone and VM

We evaluated the interaction between phone and VM through four cases. The execution time was measured in all cases. First, we measured the time cost for each sub steps in the service composition procedure. Then, we measured the overall time cost for different filter size. Last, we adjusted the network configuration to see how the execution changes.

Service composition

This experiment measures service invocation time. This time is measured on both phone and on VM. On the phone side, the service invocation time consists of three parts: image capture time, image process time, and phone marshalling time. The image capture time is the camera operation time decided by the camera hardware, which is around 422 milliseconds. The image fetching time on phone is the time for the Android system to fetch the image and pass it to the POEM bundle, which is about 196 milliseconds. The phone marshalling time is spent on coding the image and preparing the image for transmission, which is around 488 milliseconds.

On the VM side, the service consumption time consists of two parts: cloud marshalling time and image processing time. After the VM receives the data, it has to decode it and construct the image, which is cloud marshalling time. The image processing time is the time to calculate the Bilateral filter, which is around 47 milliseconds for the default configuration.

Besides the time spent on phone and on VM, the data exchange on network also costs time. It is a bit more than 4 seconds, which is dominant in the whole

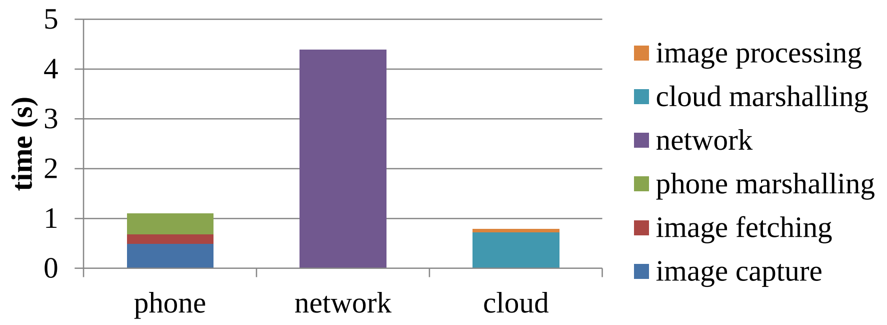


Figure 5.2: Execution Time of Each Step of Service Composition.

service composition process. Adding time for all the steps, the total time for service composition is around 7 seconds. The result is shown in Figure 5.2.

Computation offloading

The computation offloading consists of two phases. First, the bundle is migrated to the phone. The second phase is the same as service composition, where the cloud can call the service on the phone. The first phase happens only once, then the second phase service composition may happen multiple times. Thus, the bundle migration time does not contribute much in the long run.

This experiment measures PF migration time of the first phase. PF migration time period starts when service migration command is issued and ends when proxy for migrated service is available. The PF migration time for the image processing bundle is about 5 seconds. The migration time is mainly the time of transferring PF bundles on the network, which is determined by the bundle size. Different bundles may need different time.

Adjust computation

We measure the total time cost of two indicators in this scenario as shown in Figure 5.3. The 'service composition' series indicates the time cost of remote service

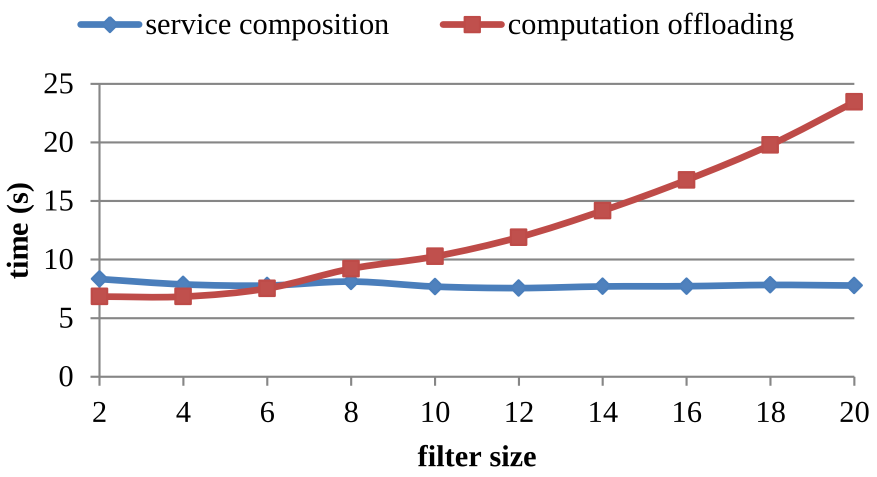


Figure 5.3: Execution Time vs. Computation Input Parameter for Service Composition and Computation Offloading.

composition, which means the image is transferred from phone to the VM and all the computation happens on the VM locally. The Figure 5.3 shows the 'service composition' series is almost a horizontal line. The time spend for the image process in the VM, which grow from 5 milliseconds to 183 milliseconds while the filter size is from 2 to 20. The 'computation offloading' series shows the time cost for migrating the image processing to the phone where the data is located to save image transfer time cost. Compared to the 'service composition' series, the 'computation offloading' series grows drastically, and the image processing time on the phone grows from 509 milliseconds to $17,847$ milliseconds on average along with the filter size increases from 2 to 20. Since the VM in the cloud is much more powerful than the smartphone, the image processing time cost on the same filter size parameter is much less than the time spent for phone local execution.

The 'service composition' time consists of two parts: the cloud image processing time which is almost constant and the rest which is constant along with the input parameter grows. The 'computation offloading' time also consists of two parts: the phone image processing time which is growing drastically and the rest which is constant along with the input parameter grows. Comparing the components of the two

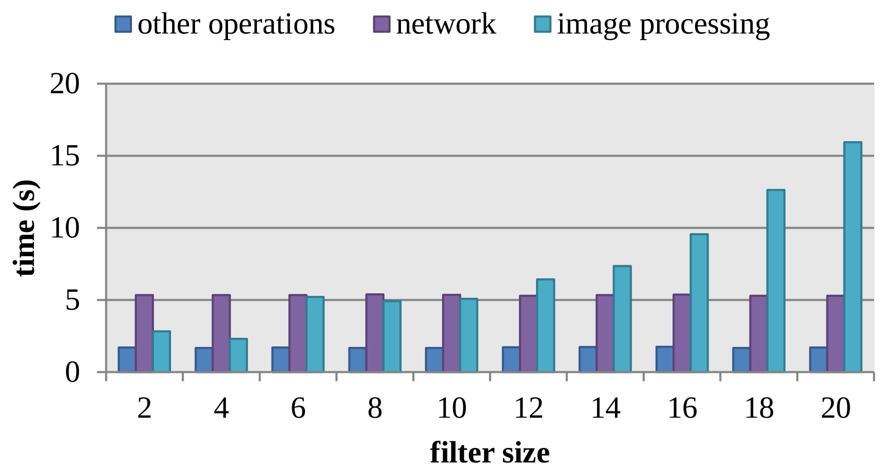


Figure 5.4: Execution Time Components of Computation Offloading vs. Computation Input Parameter.

series, they both contains the image processing part where the difference is one on phone and the other on VM. The cloud image processing time can be simply ignored compared to the phone image processing time. Figure 5.4 shows the components of the offloading execution time. The image processing time increases drastically since the phone processor power is limited and the input parameter drives the execution time increase. The time spent on network keeps constant because the data amount transferred over network maintains same. The time spent on other miscellaneous operations remains constant since the operations are same, mainly including the camera operations. The constant part of the two series are different: ‘service composition’ series transfers the image which is large amount of data, while ‘computation offloading’ series transfers the small amount of data that can be ignored compared to the image data size.

The intersection of execution time on phone and WiFi offloading is the Boundary input value (BIV) [Kosta *et al.*, 2012] that shows the offloading benefit starting point. The Figure 5.3 shows the filter size 6 is the BIV for the interaction between phone and VM. If the application inputs filter size greater than 6, the service composition is the better choice; while the computation offloading is better for the other cases.

Adjust network

To study the network impact on the system, we configured the WiFi router to the low bandwidth to mimic the real scenario on the road. The Figure 5.5 shows the execution time of service composition and computation offloading under various network configurations. We chose 7k bps and 1k bps network bandwidth configurations in the testing. The results showed that both service composition and computation offloading time increase. The incremental amount of service composition is larger than the incremental amount of computation offloading time. The network status change has larger impact on the service composition than on the computation offloading, since the service composition transfers the image data to the cloud and the transferred data amount is larger than the data amount transferred by computation offloading.

When the network bandwidth decreases, the BIV increases from 6 in default network configuration to 12 in 7k bps bandwidth network configuration, and to 20 in 1k kps bandwidth network configuration. This observation suggests that computation offloading is preferred in more circumstances when the network status goes worse.

5.1.3 Interaction Between Phone and Phone

We evaluated the interaction between phone and phone in the similar way as we did for interaction between phone and VM. We measured the service composition time and computation offloading time against input filter size under various network configurations, shown in Figure 5.6. Since the two phones are the same in the experiment, so the image processing time on the both phones are the same. The service composition transfers larger amount data than the computation offloading, thus the service composition costs more time. When the network bandwidth decreases, the incremental service composition time is larger, which is similar to the observation in

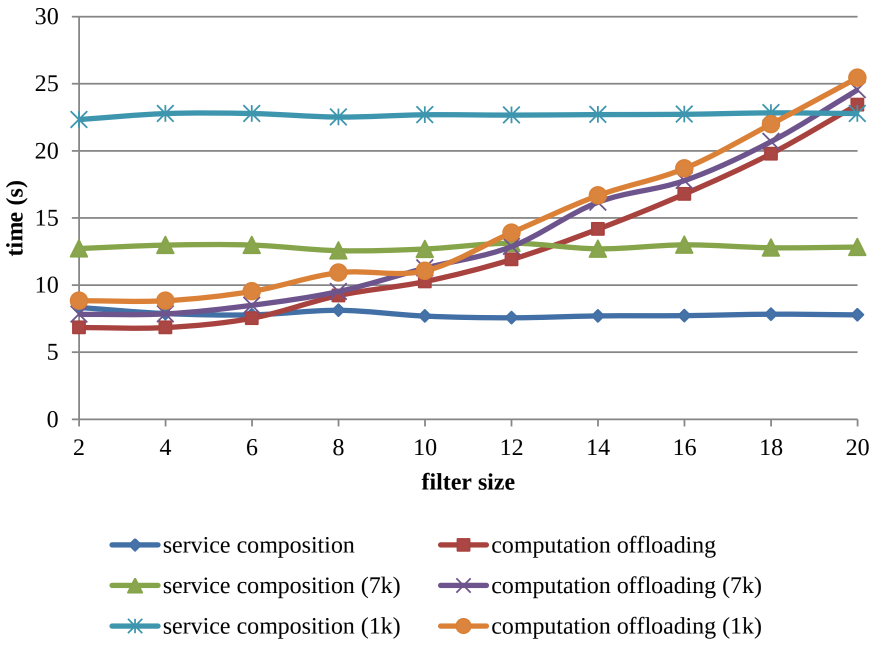


Figure 5.5: Execution Time vs. Computation Input Parameter for Service Composition and Computation Offloading Under Various Network Configurations.

the interaction of phone and VM experiments. In the experiments, the computation offloading is always better than the service composition since the it transfers less data and the service composition does get any image processing time benefit as VM does.

5.2 MIDAS Applications

The mobile cloud group developed several applications for MIDAS project, including data collection and visualization, and real time data streaming.

5.2.1 Data Collection and Visualization in Google Map

Each PICT device collects the vehicle Global Positioning System (GPS) location information as well as the velocity and acceleration information. The collected vehicle information is sent to the cloud who assembles all information and provides visualization in the Google Map. In the Figure 5.7, each red dot represents a vehicle which runs on the I-10 highway in Phoenix metropolitan, Arizona.

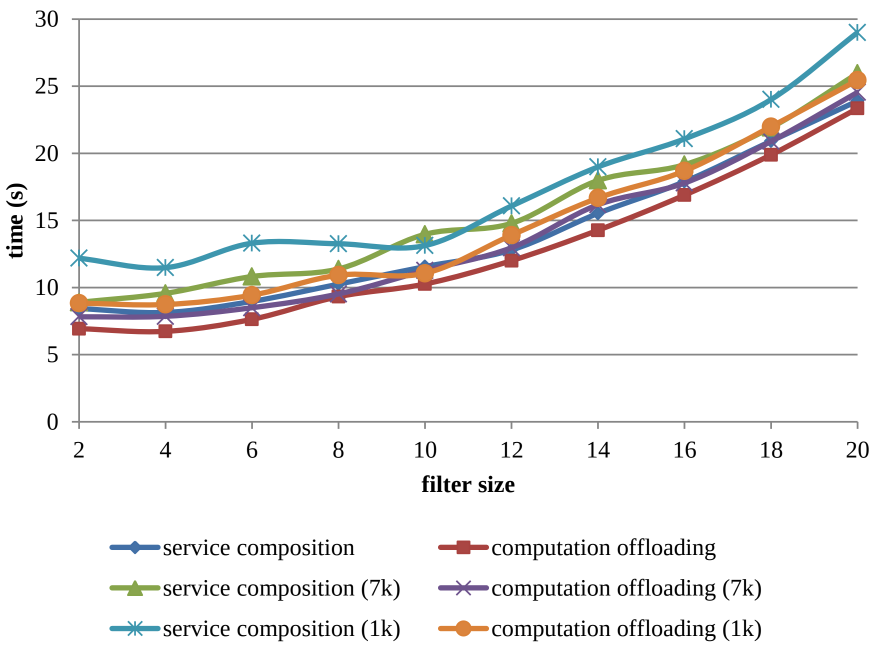


Figure 5.6: Execution Time of Interaction Between Phone and Phone vs. Computation Input Parameter for Service Composition and Computation Offloading Under Various Network Configurations.

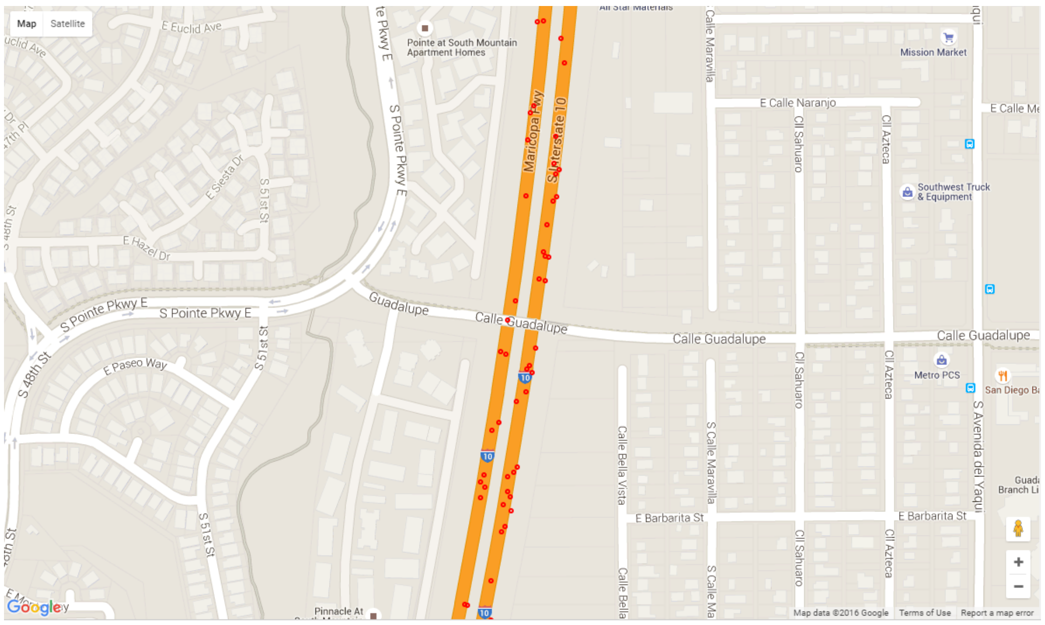
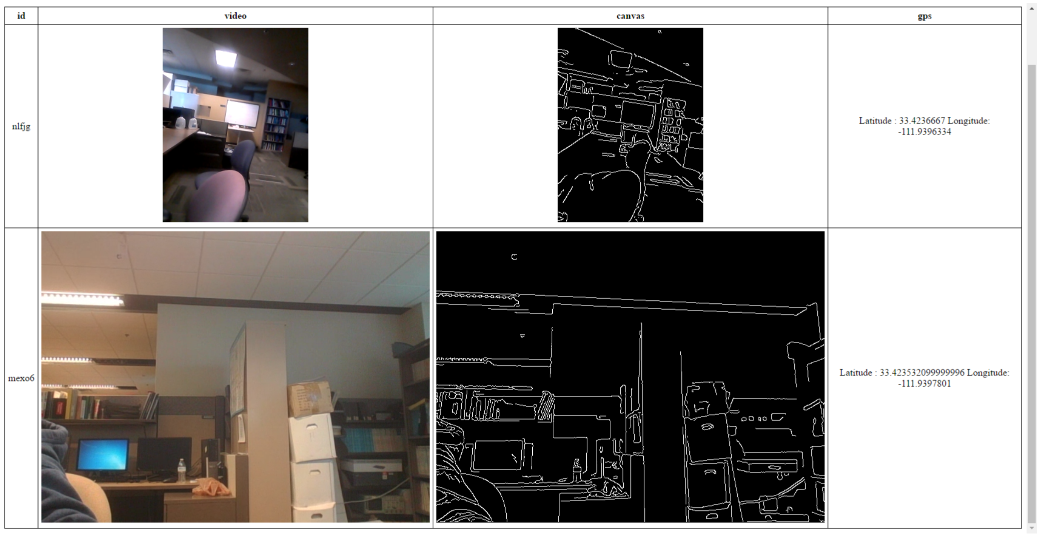


Figure 5.7: MIDAS Application: Data Collection and Visualization on Google Map.



id	video	canvas	gps
nlfjg			Latitude : 33.4236667 Longitude: -111.9396334
mexo6			Latitude : 33.423532099999996 Longitude: -111.9397801

Figure 5.8: MIDAS Application: Real Time Data Streaming from PICT Devices.

5.2.2 Real Time Data Streaming from PICT Devices

Besides location information, PICT devices would like to share their video streams captured by their cameras. The image processing function can be applied to the video streams. Figure 5.8 shows the video streams collected in the cloud as well as the line detection results.

5.2.3 Real Time Video Stream Processing System

The cloud accepts the video streams from the PICT devices, then the cloud process the video stream to fetch the information like lanes, vehicles and signposts. The stream processing module or the image processing module should be pluggable, which means the analysis module can be replaced or added to adjust the system intelligence. The system architecture is shown in Figure 5.8.

The system consists of four layers. From the stream source to stream sink, they are: input layer, front-end streaming layer, pluggable analysis layer and sink broadcasting layer. The input layer accepts streams from PICT devices or generates streams from historical video files. The source streaming servers and sink streaming servers

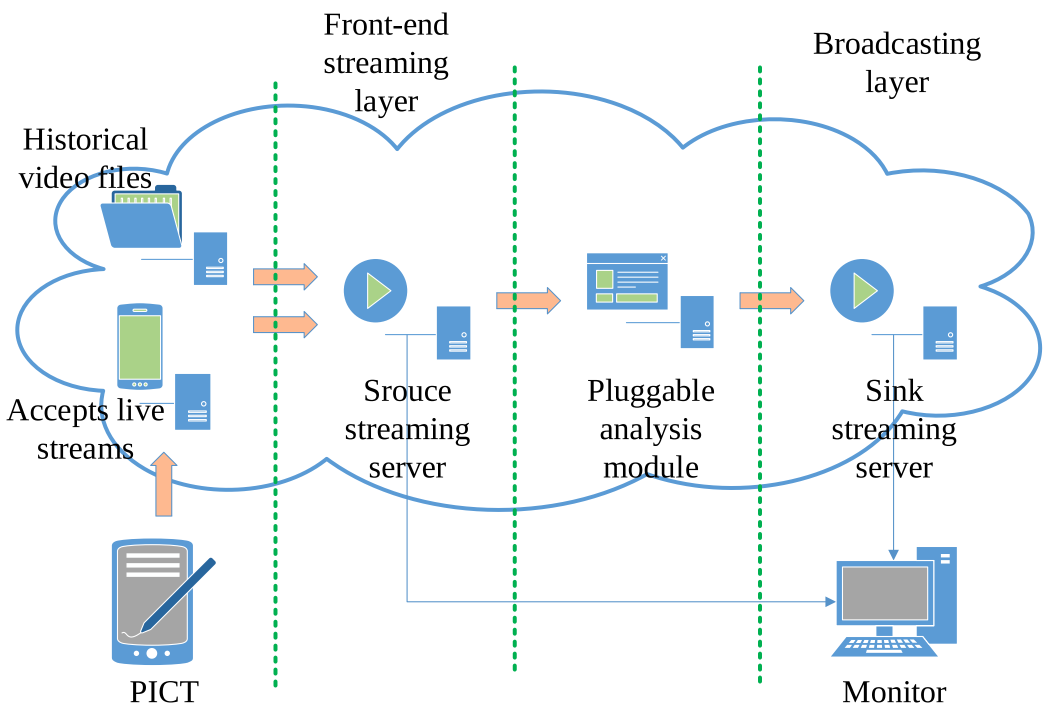


Figure 5.9: MIDAS Application: Real Time Video Stream Processing.

are media server clusters to host the video streams. The output of both streaming servers are video streams. The pluggable analysis modules consumes the streams from the source streaming servers and push the process results to the sink streaming servers. The monitor site fetches the streams from the source and sink streaming servers to see the system status and processing results.

Chapter 6

CONCLUSION AND FUTURE WORK

This chapter concludes the dissertation by summarizing the contributions of the work and highlighting the future directions.

6.1 Conclusion

In this dissertation, I proposed a novel application running platform, called POEM,for mobile cloud computing that allows mobile users to offload and compose mobile cloud application with little management overhead. The implementation is based on OSGi platform and XMPP protocols. The proposed service platform handles service migration, service discovery and service composition seamlessly in a transparent fashion. The evaluation showed the proposed service platform is flexible and efficient.

Three offloading strategies were proposed to solve the offloading decision making problem in various scenarios. The first strategy solves the devision making in unstable network environment. The proposed solution modeled unstable network as an alternating renewal process. The execution time and energy consumption are analyzed in this unstable network scenario. The offloading problem is formulated as an optimization problem to find an application partition configuration that can provide offloading benefit when low network availability is low. A bee colony based algorithm was proposed to calculate the application partition resistant to network unavailability. And a bayesian decision approach was proposed to validate the partition and output the final offloading decision. The simulation results demonstrated good performance of proposed solution.

The second strategy solves multiple objective offloading decision making problem. The proposed a multi-factor multi-site risk-based offloading model abstracts the offloading impact factors as for offloading benefit and offloading risk. The offloading decision is made based on a comprehensive offloading risk evaluation. This proposed model is generic and expendable. Four offloading impact factors are presented to show the construction and operation of the presented offloading model, which can be easily extended to incorporate more factors to make offloading decision more comprehensive. The overall offloading benefits and risks are aggregated based on the mobile cloud users' preference. An ant-based algorithm is proposed to calculate the assignment from application components to surrogate sites. The performance evaluation presents the practicality of the presented solution.

The third strategy solves the offloading decision making for a series. The proposed a service composition topology reconfiguration model for multi-site service composition application abstracts the service composition topology reconfiguration as five-element tuple. The proposed model deals with a series of decision points rather than one-time decision. It also defined the surrogate site states and topology reconfiguration actions. Moreover, it uses the mobile device energy as an example to illustrate the service composition objective and the reconfiguration rewards. Based on the proposed model, we proposed three algorithms to solve the reconfiguration problem. Each algorithm is suit for one application scenario: (a) the finite horizon algorithm assumes the prior knowledge of application execution time; (b) the infinite algorithm generalizes the assume of finite algorithm to the infinity, which eliminates the restriction of the prior knowledge assumption; and (c) the third algorithm is motivated by the real world multi-site service composition applications that involves many aspect leading to large state space. The evaluation results demonstrated the applicability and good properties of all three algorithms.

6.2 Future Work

Mobile cloud is still in its early stages of development and an active area of exploration. Below we present some promising research directions:

- Augmented reality with mobile cloud assistant: The mobile devices enables unlimited applications in the life. The augmented reality is one of the hottest topic in both academy and industry. The augmented reality requires lots of information and computation, which may not be available on the mobile device itself. Thus the cloud is necessary to aggregate the information, compute the augmented environment, and send to the mobile devices. Some augmentation functions that are computation intensive may be offloaded to the cloud and personalization may be applied to the computation. The offloading in such augmented reality scenario needs further study to make proper offloading decision.

- Personal assistant: Besides the augmented reality in the daily life, the personal assistant is another hot topic. The personal assistant on mobile device needs high intelligence to interact with human. This type of intelligence at present is only available through large scale machine learning system in the cloud. Thus, the personal assistant on the mobile device has to contact the cloud for the interaction with human. In addition to the information retrieval, the cloud that acts like the central brain may offload some tasks to the mobile devices to act directly to assist human life, which is exactly like the experiments in the Chapter 5.

- Scale offloading/composition in mobile cloud: The current computation offloading/composition exists in the small group of resources, such as the resources acquired by particular user. However, scalability creates benefit. Large group of offloading/composition introduces problems not only in the resource scheduling and management, but also the privacy and security, on-time system response, and risk management as well. The

- Internet of Things with mobile cloud: The emerging of IoT brings challenges for the mobile cloud. Since many devices carry computation unit, which may be mobile or stationary, the collaboration and management of these devices and cloud become challenging. The ad-hoc mobile cloud resources are dynamic. The devices may join to leave at any time. More research is in need to manage this type of dynamic cloud.

REFERENCES

Abebe, E. and C. Ryan, "A hybrid granularity graph for improving adaptive application partitioning efficacy in mobile computing environments", in "10th IEEE International Symposium on Network Computing and Applications (NCA)", pp. 59–66 (IEEE, 2011).

Abebe, E. and C. Ryan, "Adaptive application offloading using distributed abstract class graphs in mobile environments", Journal of Systems and Software **85**, 12, 2755–2769 (2012).

Alazawi, Z., S. Altowaijri, R. Mehmood and M. B. Abdljabar, "Intelligent disaster management system based on cloud-enabled vehicular networks", in "11th International Conference on ITS Telecommunications (ITST)", pp. 361–368 (IEEE, 2011).

Arif, S., S. Olariu, J. Wang, G. Yan, W. Yang and I. Khalil, "Datacenter at the airport: Reasoning about time-dependent parking lot occupancy", IEEE Transactions on Parallel and Distributed Systems **23**, 11, 2067–2080 (2012).

Booker, G., A. Sprintson, C. Singh and S. Guikema, "Efficient availability evaluation for transport backbone networks", in "International Conference on Optical Network Design and Modeling (ONDM)", pp. 1–6 (IEEE, 2008).

Bradski, G. and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library* (O'Reilly Media, Inc., 2008).

Chakareski, J., "Adaptive multiview video streaming: challenges and opportunities", Communications Magazine, IEEE **51**, 5, 94–100 (2013).

Chen, M., J. Chen and T. Chang, "Android/osgi-based vehicular network management system", Computer Communications **34**, 169–183 (2011).

Chen, M., Y. Zhang, Y. Li, S. Mao and V. Leung, "Emc: Emotion-aware mobile cloud computing in 5g", IEEE Network **29**, 2, 32–38 (2015).

Chun, B. G., 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", pp. 301–314 (2011).

Corradi, A., M. Fanelli and L. Foschini, "Vm consolidation: a real case based on openstack cloud", Future Generation Computer Systems **32**, 118–127 (2014).

Cuervo, E., 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", pp. 49–62 (ACM, 2010).

Eckel, B., *Thinking in Java* (Prentice Hall, 2006), 4 edn.

Eltoweissy, M., S. Olariu and M. Younis, “Towards autonomous vehicular clouds”, in “International Conference on Ad Hoc Networks”, pp. 1–16 (Springer, 2010).

Fielding, R. T. and R. N. Taylor, “Principled design of the modern web architecture”, ACM Transactions on Internet Technology (TOIT) **2**, 2, 115–150 (2002).

Flinn, J., “Cyber foraging: Bridging mobile and cloud computing”, Synthesis Lectures on Mobile and Pervasive Computing **7**, 2, 1–103 (2012).

Gédéon, W. J., *OSGi and Apache Felix 3.0 Beginner’s Guide* (Packt Publishing Ltd, 2010).

Gerla, M., “Vehicular cloud computing”, in “2012 The 11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)”, pp. 152–155 (2012).

Giurgiu, I., O. Riva and G. Alonso, “Dynamic software deployment from clouds to mobile devices”, in “Middleware”, pp. 394–414 (Springer, 2012).

Giurgiu, I., O. Riva, D. Juric, I. Krivulev and G. Alonso, “Calling the cloud: Enabling mobile phones as interfaces to cloud applications”, Middleware pp. 83–102 (2009).

Hildebrand, J., P. Millard, R. Eatmon and P. Saint-Andre, *XEP-0030: Service Discovery*, XMPP Standards Foundation (XSF) (2008).

Hsu, C.-Y., C.-S. Yang, L.-C. Yu, C.-F. Lin, H.-H. Yao, D.-Y. Chen, K. R. Lai and P.-C. Chang, “Development of a cloud-based service framework for energy conservation in a sustainable intelligent transportation system”, International Journal of Production Economics (2014).

Huang, D., “Mobile cloud computing”, IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter **6**, 10, 27–31 (2011).

Huang, D., T. Xing and H. Wu, “Mobile cloud computing service models: a user-centric approach”, IEEE Network **27**, 5, 6–11 (2013a).

Huang, D., T. Xing and H. Wu, “Mobile cloud computing service models: A user-centric approach”, IEEE Networks **27**, 5, 6–11 (2013b).

Huang, D., X. Zhang, M. Kang and J. Luo, “Mobicloud: Building secure cloud framework for mobile computing and communication”, in “2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)”, pp. 27–34 (Nanjing, 2010).

Huang, D., Z. Zhou, L. Xu, T. Xing and Y. Zhong, “Secure data processing framework for mobile cloud computing”, in “IEEE INFOCOM’s Workshop on Cloud Computing”, (2011).

Huerta-Canepa, G. and D. Lee, “A virtual cloud computing provider for mobile devices”, in “Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond”, p. 6 (ACM, 2010).

Jereb, L., “Network reliability: models, measures and analysis”, in “Proceedings of the 6th IFIP Workshop on Performance Modelling and Evaluation of ATM Networks, Tutorial Papers, Ilkley, UK”, (1998).

Jiang, D. and L. Delgrossi, “Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments”, in “Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE”, pp. 2036–2040 (IEEE, 2008).

Jung, W., 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”, pp. 353–362 (ACM, 2012).

Karaboga, D. and B. Akay, “A comparative study of artificial bee colony algorithm”, Applied Mathematics and Computation **214**, 1, 108–132 (2009).

Kemp, R., N. Palmer, T. Kielmann and H. Bal, “Cuckoo: a computation offloading framework for smartphones”, in “Mobile Computing, Applications, and Services”, pp. 59–79 (Springer, 2012).

Kim, K.-H., S.-J. Lee and P. Congdon, “On cloud-centric network architecture for multi-dimensional mobility”, ACM SIGCOMM Computer Communication Review **42**, 4, 509–514 (2012).

Kim, W. and M. Gerla, “Navopt: Navigator assisted vehicular route optimizer”, in “Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)”, pp. 450–455 (IEEE, 2011).

Kosta, S., A. Aucinas, P. Hui, R. Mortier and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading”, in “2012 Proceedings IEEE INFOCOM”, pp. 945–953 (2012).

Kovachev, D., “Framework for computation offloading in mobile cloud computing”, IJIMAI **1**, 7, 6–15 (2012).

Kristensen, M. D., “Scavenger: Transparent development of efficient cyber foraging applications”, in “IEEE International Conference on Pervasive Computing and Communications (PerCom)”, pp. 217–226 (2010).

Levendovszky, J., L. Jereb, Z. Elek and G. Vesztergombi, “Adaptive statistical algorithms in network reliability analysis”, Performance Evaluation **48**, 1, 225–236 (2002).

Lin, J. and C. Dyer, “Data-intensive text processing with mapreduce”, Synthesis Lectures on Human Language Technologies **3**, 1, 1–177 (2010).

Liu, K., *Applied Markov Decision Processes* (Tsinghua University Press, 2004).

Liu, Y., “An energy-efficient multisite offloading algorithm for mobile devices”, International Journal of Distributed Sensor Networks **2013** (2013).

Ma, R., K. Lam, C. Wang and C. Zhang, "A stack-on-demand execution model for elastic computing", in "Proc. of the 39th International Conference on Parallel Processing (ICPP 2010)", pp. 208–217 (2010).

Ma, R. K., K. T. Lam and C.-L. Wang, "excloud: Transparent runtime support for scaling mobile applications in cloud", in "International Conference on Cloud and Service Computing (CSC)", pp. 103–110 (IEEE, 2011).

March, V., Y. Gu, E. Leonardi, G. Goh, M. Kirchberg and B. S. Lee, "μcloud: towards a new paradigm of rich mobile applications", Procedia Computer Science **5**, 618–624 (2011).

McCullough, J. C., Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization", in "USENIX Annual Technical Conf", (2011).

Millard, P., P. Saint-Andre and R. Meijer, *XEP-0060: Publish-Subscribe*, XMPP Standards Foundation (XSF) (2010).

Mittal, R., A. Kansal and R. Chandra, "Empowering developers to estimate app energy consumption", in "Proceedings of the 18th annual international conference on Mobile computing and networking", pp. 317–328 (ACM, 2012).

Ni, Q., E. Bertino and J. Lobo, "Risk-based access control systems built on fuzzy inferences", in "Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security", pp. 250–260 (ACM, 2010).

Niu, J., W. Song and M. Atiquzzaman, "Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications", Journal of Network and Computer Applications **37**, 334–347 (2014).

Olariu, S., T. Hristov and G. Yan, "The next paradigm shift: from vehicular networks to vehicular clouds", Mobile Ad Hoc Networking: Cutting Edge Directions, Second Edition pp. 645–700 (2013).

OSGi-Alliance, *OSGi Core Release 5* (2012).

Ou, S., Y. Wu, K. Yang and B. Zhou, "Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments", in "IEEE International Conference on Communications (ICC)", pp. 1856–1860 (IEEE, 2008).

Ou, S., K. Yang and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems", in "Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)", (IEEE, 2006).

Papakos, P., L. Capra and D. S. Rosenblum, "Volare: context-aware adaptive cloud service discovery for mobile systems", in "Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware", pp. 32–38 (2010).

Pham-Gia, T. and N. Turkkan, "System availability in a gamma alternating renewal process", Naval Research Logistics (NRL) **46**, 7, 822–844 (1999).

Pitkänen, M., T. Kärkkäinen, J. Ott, M. Conti, A. Passarella, S. Giordano, D. Puccinelli, F. Legendre, S. Trifunovic, K. Hummel *et al.*, "Scampi: Service platform for social aware mobile and pervasive computing", ACM SIGCOMM Computer Communication Review **42**, 4, 503–508 (2012).

Ra, M.-R., B. Priyantha, A. Kansal and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors", in "Proceedings of the 2012 ACM Conference on Ubiquitous Computing", pp. 1–10 (ACM, 2012).

Rahimi, M. R., N. Venkatasubramanian, S. Mehrotra and A. V. Vasilakos, "Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture", in "Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing", pp. 83–90 (IEEE Computer Society, 2012).

Saint-Andre, P., "Extensible messaging and presence protocol (xmpp): Core", (2011).

Saripalli, P. and B. Walters, "Quirc: A quantitative impact and risk assessment framework for cloud security", in "IEEE 3rd International Conference on Cloud Computing (CLOUD)", pp. 280–288 (IEEE, 2010).

Shin, D., K. Kim, N. Chang, W. Lee, Y. Wang, Q. Xie and M. Pedram, "Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics", in "18th Asia and South Pacific Design Automation Conference (ASP-DAC)", pp. 59–64 (IEEE, 2013).

Sinha, K. and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading", in "IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)", pp. 184–194 (IEEE, 2011).

Smit, M., M. Shtern, B. Simmons and M. Litoiu, "Partitioning applications for hybrid and federated clouds", in "Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research", pp. 27–41 (IBM Corp., 2012).

Taillard, É. D. and L. M. Gambardella, "Adaptive memories for the quadratic assignment problem", Technical Report IDSIA-87-97 (1997).

Verbelen, T., T. Stevens, F. De Turck and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing", Future Generation Computer Systems **29**, 2, 451–459 (2013).

Wang, Y., X. Lin and M. Pedram, "A nested two stage game-based optimization framework in mobile cloud computing system.", in "SOSE", pp. 494–502 (2013).

Wolski, R., S. Gurun, C. Krintz and D. Nurmi, "Using bandwidth data to make computation offloading decisions", in "IEEE International Symposium on Parallel and Distributed Processing (IPDPS)", pp. 1–8 (IEEE, 2008).

Wu, H. and D. Huang, "Modeling multi-factor multi-site risk-based offloading for mobile cloud computing", in "10th International Conference on Network and Service Management (CNSM)", pp. 230–235 (IEEE, 2014).

Wu, H. and D. Huang, "Mosec: Mobile-cloud service composition", in "3rd International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)", (IEEE, 2015).

Wu, H., D. Huang and S. Bouzefrane, "Making offloading decisions resistant to network unavailability for mobile cloud collaboration", in "the 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)", (IEEE, 2013).

Wu, H., D. Huang and Y. Zhu, "Establishing a personal on-demand execution environment for mobile cloud applications", Mobile Networks and Applications **20**, 3, 297–307 (2015).

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

Xu, Q., R. Segupta, D. Jiang and D. Chrysler, "Design and analysis of highway safety communication protocol in 5.9 ghz dedicated short range communication spectrum", in "The 57th IEEE Semiannual Vehicular Technology Conference", vol. 4, pp. 2451–2455 (IEEE, 2003).

Yang, L., J. Cao, Y. Yuan, T. Li, A. Han and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing", ACM SIGMETRICS Performance Evaluation Review **40**, 4, 23–32 (2013).

Yang, X., L. Liu, N. H. Vaidya and F. Zhao, "A vehicle-to-vehicle communication protocol for cooperative collision warning", in "The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services", pp. 114–123 (IEEE, 2004).

Yoon, C., D. Kim, W. Jung, C. Kang and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring", in "USENIX ATC", (2012).

Zhang, L., 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", pp. 105–114 (ACM, 2010a).

Zhang, X., S. Jeong, A. Kunjithapatham and S. Gibbs, "Towards an elastic application model for augmenting computing capabilities of mobile platforms", in "Mobile wireless middleware, operating systems, and applications", pp. 161–174 (Springer, 2010b).

Zhang, X., A. Kunjithapatham, S. Jeong and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing", Mobile Networks and Applications **16**, 3, 270–284 (2011).