THE HONG KONG POLYTECHNIC UNIVERSITY
DEPARTMENT OF COMPUTING

COMPUTATION PARTITIONING IN MOBILE CLOUD APPLICATIONS: MODELING, OPTIMIZATION AND EVALUATIONS

YANG Lei

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

January 2014

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

YANG Lei (Name of Student)

Abstract

The proliferation of sensors on mobile devices and ubiquitous network access to clouds enable many mobile cloud applications such as augmented reality (e.g., Google Glass), voice recognition (e.g., iPhone Siri), real time translation and so on. Computation partitioning between the mobile device and the cloud for these applications is an important and challenging research topic. Although there are works done on some aspects of this study, how to provide a systematic approach to support the partitioning for various models of applications and systems is yet to be addressed. In this thesis, we classify computation partitioning into different models by considering two dimensional properties: application dimension and system dimension. On application dimension, we do partitioning for two types of applications: computation dependent application and computation independent application. On system dimension, we do the partitioning for two types of systems: single user system and multiple users system. In this thesis, we focus on the study of three models: 1) computation independent application and single user system, named as *computation offloading*, 2) computation dependent application and single user system, named as *single user computation partitioning*, 3) computation dependent application and multiple user system, named as *multiple user computation partitioning*. The thesis contains three parts, which tackle the most urgent and significant issues in terms of the thee models. The details are as follows.

First, we study the simplest model of computation partitioning, where the application is composed of *independent* computations and the partitioning decision is done for one single user. We take the RFID tracking as a case study, and demonstrate that computation partitioning can significantly improve the application performance. In particular, we consider the RFID system that attaches the RFID reader on the moving object, and deploys passive RFID tags in the environment. The moving object collects the noisy RFID readings, and perform continuous estimation of its position in real time. Traditional approaches such as Particle Filter (PF) can achieve high accuracy, but require a large amount of computations

on the device. The approaches are hard to be implemented on some mobile devices that are constrained by the computing capabilities and battery. Other existing approaches such as Weighted Centroid Localization (WCL) are cheap in computational cost, but yield bad accuracy especially when the object's speed is high. Thus, we propose an adaptive approach to achieve accuracy and energy efficiency. Our approach can adaptively choose costly PF and cheap WCL according to the estimated speed of the object, and adaptively partition the computations between the mobile device and infrastructure servers or clouds based on the quality of network connections. We evaluate our solution through real world experiments, and show that our proposed approach with computation partitioning outperforms other approaches in terms of both accuracy and efficiency.

Second, we study the model of single user computation partitioning, where the application is composed of *dependent* computations and the partitioning decision is done for one single user. We found that most existing works on computation partitioning pertains to this model. We tackle two issues that are important but not solved in existing works. *One issue* is partitioning of data streaming application. In this work, we aim at optimizing the partition of a data stream application between mobile and cloud so that the application has maximum speed/throughput in processing the streaming data. To the best of our knowledge, ours is the first work to study 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 in other applications. We propose a framework to provide runtime support for the dynamic partitioning and execution of the application. Different from existing works, the framework not only allows the dynamic partitioning for a single user but also supports the multiple tenancy service invocation in the cloud to achieve efficient utilization of the underlying cloud resources. The framework is designed on the elastic cloud fabrics for better scalability. The optimization of the partitioning for each single user is by using genetic algorithm. Through both extensive simulations and experiments on real world applications, we show that our method can achieve more than 2X better performance over the execution without partitioning.

The other issue is computation partitioning under dynamic mobile cloud environments. Existing works assume that the computational and data transmission cost of each part of the application remains the same as the application is running. This assumption does not hold in dynamic mobile cloud environments, where the device and network connection status may fluctuate, and thus affects the computational and transmission cost. In this case,

the one time partitioning of the application may yield significant performance degradations. Therefore, we consider updating the partition periodically during the course of application execution, which is named as computation repartitioning in our thesis. We propose a framework for run time computation repartitioning in dynamic mobile cloud environments. Based on this framework, we take the dynamic network connection to clouds as a case study, and design an online solution, Foreseer, to solve the mobile cloud application repartitioning problem. We evaluate our solution based on real world data traces that are collected in a campus WiFi hotspot testbed. The result shows that our method can achieve significantly shorter completion time over previous approaches.

Third, we study the most complex model of computation partitioning, where the application is composed of dependent computations, and the partitioning decision is made for multiple users. In this model, the multiple users compete for the computing resources shared by the users on the cloud. The users' partitioning decisions are dependent with each other. To achieve high system performance, the users' partitioning decisions should be considered jointly with the scheduling of computations on the shared cloud resources. To the best of our knowledge, we are the first to study the Multiple user Computation Partitioning Problem (MCPP). We show that MCPP is different from and more difficult than the classical job scheduling problems. In classical job scheduling problems, the computations are allowed to schedule onto arbitrary resources including the mobile devices and cloud resources, while in MCPP the users computations can not be scheduled to other usersdevices. We design an offline heuristic algorithm, namely *SearchAdjust*, to solve MCPP. We demonstrate through benchmarks that *SearchAdjust* outperforms the classical job scheduling approaches by 10% on average in terms of application delay. Based on *SearchAdjust*, we also design an online algorithm for MCPP that can be easily deployed in practical systems. We validate the effectiveness of our online algorithm using real world load traces.

Publications

Journal Paper

1. **Lei Yang**, Jiannong Cao, Hui Cheng, and Yusheng Ji, "*Multi-user Computation Partitioning for Latency Sensitive Mobile Cloud Applications*", submitted to IEEE Transaction on Computers (TC)

2. **Lei Yang**, Jiannong Cao, Weiping Zhu, Shaojie Tang, "*Accurate and Efficient Object Tracking based on Passive RFID*", submitted to IEEE Transaction on Mobile Computing (TMC)

3. **Lei Yang**, Jiannong Cao, Shaojie Tang, Di Han, and Neeraj Suri, "*Run Time Application Repartitioning in Dynamic Mobile Cloud Environments*", submitted to IEEE Transaction on Cloud Computing (TCC)

4. Weiping Zhu, Jiannong Cao, Yi Xu, **Lei Yang**, and Junjun Kong, "*Fault-Tolerant RFID Reader Localization Based on Passive RFID Tags*", accepted by IEEE Transaction on Parallel and Distributed Systems (TPDS)

5. Chao Ma, Jiannong Cao, **Lei Yang**, Jun Ma, and Yanxiang He, "*Effective Social Relationship Measurement based on User Trajectory Analysis*", Journal of Ambient Intelligence and Humanized Computing (JAIHC), Vol.5, No.1, pp.39-50, 2014

6. **Lei Yang**, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, Alvin Chan "*A Framework for Partitioning and Execution of Data Stream Application in Mobile Cloud Computing*", ACM SigMetrics Performance Evaluation Review (PER), Vol.40, No.4, pp.23-32, March, 2013

7. **Lei Yang**, Jiannong Cao, "*Computation Partitioning in Mobile Cloud Computing: A Survey*", ZTE Communication. Vol.11, No.4, pp.08-17, Dec., 2013

Conference Paper

1. Mike Jia, Jiannong Cao, **Lei Yang**, "*Optimal Offloading of Concurrent Tasks for Computation-Intensive Applications in Mobile Cloud Computing*", accepted by INFOCOM 2014 Workshop on Mobile Cloud Computing

2. **Lei Yang**, Jiannong Cao, Shaojie Tang, Tao Li, and Alvin Chan, "*A Framework for Partitioning and Execution of Data Stream Application in Mobile Cloud Computing*", in Proc. of IEEE International Conference on Cloud Computing (CLOUD), pp.794-802, 2012

3. **Lei Yang**, Jiannong Cao, Weiping Zhu, and Shaojie Tang, "*A Hybrid Method for achieving High Accuracy and Efficiency in Object Tracking using Passive RFID*", in Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom), pp.109-115, 2012

4. Weiping Zhu, Jiannong Cao, Yi Xu, **Lei Yang**, and Junjun Kong, "*Fault-Tolerant RFID Reader Localization Based on Passive RFID Tags*", in Proc. of IEEE International Conference on Computer Communications (INFOCOM), pp.2183-2191, 2012

5. Chao Ma, Jiannong Cao, **Lei Yang**, Jun Ma, Tao Li, and Junjun Kong, "*An Approach to Measuring User Relationship Based on Location Trail*", in Proc. of Joint Conference on Hormonious Human Man Evironment (HHME), pp.146-153, 2011

Acknowledgements

Foremost, I would like to express my deepest gratitude to my supervisor Prof. Jiannong Cao for his systematical training of my Ph.D study and research. He has immense knowledge, and show great enthusiasm to pursue impacts in research. I learn a lot from him not only on how to think, write, and present, but also on how to do things carefully and quickly. Besides an outstanding researcher, he is also an excellent leader and a good person. His talent in leadership and strong personality influence me in many aspects of my life. Working with him is the most beneficial and valuable experience in my life.

I would like to thank my wife Junjie Hou for her endless love during my days to pursue the Ph.D degree. In most time, we were not staying together, but she was always there, listening to my experiences, encouraging me and dreaming beautiful days with me. It was her who takes away loneliness from and brings precious happiness to my Ph.D life.

My sincere thanks also goes to Prof. Neeraj Suri for offering me opportunity of four months visiting research at Technical University Darmstadt (TUD), Germany. I thank the colleagues Stefan, Habib, Tsveti, Thorsten, Daniel, Hamza, Jesus and all other members in Deeds group at TUD, for their help in my research and life during my visiting.

I thank my colleagues Xuefeng Liu, Vaskar Raychoudhury, Joanna Siebert, Weiping Zhu, Tao Li, Chisheng Zhang, Yang Liu, Gang Yao, Wei Feng, Jingjing Li, Jie Zhou, Miao Xiong, Junjun Kong, Chao Yang, Xin Xiao, Liang Yang, Chao Ma, Jun Ma, Yin Yuan, Zongjian He, Guanqing Liang, Peng Guo, Yaguang Huangfu, Wanyu Lin, Junhao Zheng, Rui Liu and all other members of our research group at Hong Kong PolyU that I cannot

enumerate here. We discuss research ideas, and share happiness and sadness in our daily lives. Thanks for their helps.

Last but not least, I would like to thank my parents and my elder brother. Their love is the most powerful strength that drives me to move forward.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADR: Application Delay Ratio

AP: Access Point

AR: Augmented Reality

CCR: Communication to Computation Ratio

DAG: Directed Acyclic Graph

DF: data flow

FBP: Flow Based Programming

HEFT: Heterogeneous Earliest Finish Time

HFS: Hybrid Flow Shop

LS: List Scheduling

MCC: Mobile Cloud Computing

MCP: Maximum Clique Problem

MCPP: Multiple user Computation Partitioning Problem

MEDLS: Minimum Extra Delay List Scheduling

MILP: Mixed Integer Linear Programming

PF: Particle Filter

PRL: Performance Resource Load

RFID: Radio Frequency Identification

SCPP: Single user Computation Partitioning Problem

SLA: Service Level Agreement

SPM: Sampled Pattern Matching

TSPHC: Task Scheduling Problem in Heterogeneous Computing

WCL: Weighted Centroid Localization

WSN: Wireless Sensor Network

Chapter 1

Introduction

This research investigates the requirements and presents the modeling, optimization and evaluation of computation partitioning in mobile cloud applications. In this chapter, we first describe the background of mobile cloud computing in Section 1.1. After that, we explain the motivation of our work in Section 1.3. In Section 1.4, we summarize the main contributions of this thesis. Finally, we outline the organization of this thesis in Section 1.5.

1.1 Mobile Cloud Computing

Cloud computing is an important transition and paradigm shift in IT service delivery driven by economies of scale. It provides a computing paradigm that enables a shared pool of virtualized, dynamically configurable, and managed computing resources to be delivered on demand to customers over the Internet and other available networks. As such and with the pay-as-you-go business model, cloud computing will also lead to changes and transformation of many industries. On the other hand, with the advances in technologies of wireless communications and portable devices, mobile computing has become integrated into the fabric of our every day life. With increased mobility, users need to run stand-alone and/or to access remote mobile applications on mobile devices.

The application of cloud services in the mobile ecosystem enables a newly emerging mobile computing paradigm, namely Mobile Cloud Computing (MCC). MCC offers great opportunities for mobile service industry, allowing mobile devices to utilize the elastic resources offered by the cloud. It is predicted that MCC services will be the platforms of

choice of IT industry for the next 20 years and generate huge revenue. There exist three approaches: 1) extending the access to cloud services to mobile devices; 2) enabling mobile devices to work collaboratively as cloud resource providers; 3) developing next generation mobile applications by leveraging cloud computing technologies, e.g., by offloading the computing resources required by applications on mobile devices to the cloud so we can create applications that far exceed traditional mobile device capabilities.

In the first approach, users use mobile devices, often through web browsers, to access software/applications as services offered by cloud. The mobile cloud is most often viewed as a Software-as-a-Service (SaaS) cloud, meaning that computation and data handling are usually performed in the cloud. The second MCC approach makes use of the resource at individual mobile devices to provide a virtual mobile cloud, which is useful in an ad hoc networking environment without the access to Interne cloud [HCL10] [Mar09]. The third MCC approach uses the cloud for storage and processing for applications running on mobile devices. The mobile cloud is most considered as a Platform-as-a-Service (PaaS) cloud, which is leveraged to augment the capability of mobile devices through offloading the computation and data storage from the mobile devices [CBC10] [CIM+11].

In this thesis, we focus on the third approach, which sets the future trend and represents the major effort in research. We aim to model, design and implement mobile cloud systems that can achieve adaptive and elastic application execution between the mobile devices and the cloud. This is motivated by the fact that many mobile devices like the iPhone have been equipped with various kinds of sensors and multimedia capabilities. We foresee that a lot of new mobile applications such as multimedia applications, object recognition, location-based social networks and augmented reality, will become highly demanded. By moving the computation to the cloud, these advanced mobile applications which could not be accommodated before due to the lack of significant computing capability and energy power of mobile devices, will be enabled and enjoyed by mobile users.

1.2 Computation Offloading and Partitioning

With great benefits of the third MCC approach, we face the problem of computation partitioning which allocates the computations of the application between the mobile device and the cloud, such that the execution cost is minimized. We classify the computation partitioning by considering two dimensional properties: *application dimension* and *system dimension*. On the application dimension, we consider the dependency relationship between the computations involved in the application. We use two terminologies, *computation offloading* and *computation partitioning*, respectively in term of the partitioning of computation independent application and computation dependent application. On the system dimension, we consider whether the partitioning decision is for one single user or for multiple users. We use the terminologies, *single user computation partitioning* and *multiple user computation partitioning*, respectively in terms of the partitioning for single user and for multiple users. The detailed illustrations about the classification are as follows.

1.2.1 Computation Offloading v.s. Computation Partitioning

From the application dimension, we have two types of applications: *computation independent application*, where the application contains a set of computational modules that have no data dependence between each other, and *computation dependent application*, where the application is divided into a set of computational modules that have data dependence between each other. In computation independent application, we treat each computational module separately and make the decision for each computational module that whether it should be executed on the mobile device or on the cloud. Note that the partitioning decision of each computational module is independent with each other. In computation dependent application, there exists data communications among the computational modules. We need to make a optimal global decision for all the computational modules, that which modules should be executed on the mobile devices and which modules should be executed on the cloud. Partitioning of computation dependent application usually is more difficult than partitioning of computation independent application.

Without loss of generality, throughput the following of this thesis, the terminology *computation partitioning* particularly indicates the partitioning of computation dependent application. We use the terminology *computation offloading* to indicate the partitioning of computation independent application. There exist some applications that are not easy to be divided into multiple computational modules. The whole application is treated as one computational module. The partitioning problem is to decide whether the whole application is executed on the device or on the cloud. We classify the partitioning of these undivided applications into the category of computation offloading.

1.2.2 Single User Computation Partitioning v.s. Multiple User Computation Partitioning

From the system dimension, we have two models: *single user computation partitioning* and *multiple user computation partitioning*. In single user computation partitioning, we consider the computation partitioning problem for each single user. The partitioning decisions for the users are independent of each other. The user decides its optimal partitioning decision by itself. In multiple user computation partitioning, the users' partitioning decisions are dependent on each other due to their competition on some shared resources such as the network bandwidth, cloud computing resources and so on. The allocation of these shared resources to the users and the partitioning decisions for the users should be considered jointly. In multiple user computation partitioning, the partitioning decisions are made based on the global information from all the users, with the aim of achieving the minimum of the total execution cost of all the users. We found that most state-of-arts [CBC10] [CIM+11] [ZKJG09] [GRJ+09] [LWX09] [KAH+12] [RSM+11] [BSPO03] [YCY+13] pertain to single user computation partitioning, while the multiple user partitioning model is first studied in our recent works [YCCJ13].

Fig.1.1 illustrates the difference between single user computation partitioning and multiple user computation partitioning. In single user computation partitioning, each user makes its own partitioning decision. The optimization of partitioning can be performed at various places of the mobile cloud system, i.e., on the device (Fig.1.1a), in the wireless network

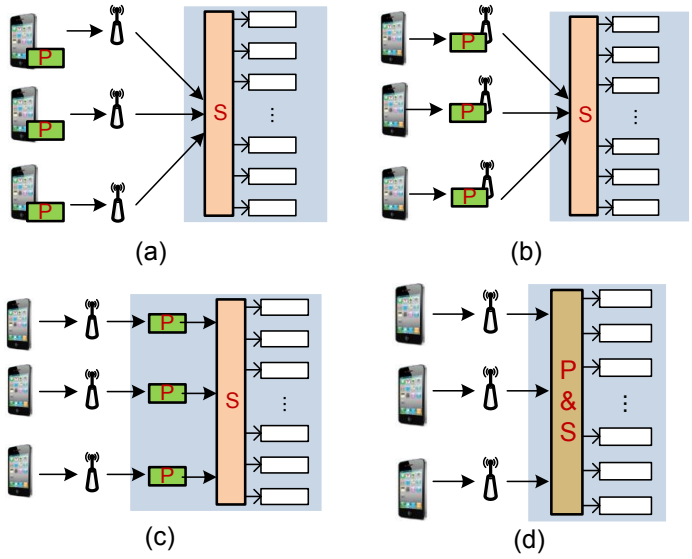


Fig. 1.1: System model: (a)(b)(c) are multiple user model, and (d) are single user model

(Fig.1.1b), and on the cloud(Fig.1.1c). Note that the block with 'P' indicates the partitioning function. The block with 'S' represents scheduling of the offloaded computations from mobile users onto the shared cloud resources.

In multiple user computation partitioning, the partitioning and scheduling functions are coupled together. The users' partitionings are dependent on each other. This model is suitable for the scenario where the users may compete for certain resources shared by the users, e.g., the servers on the cloud. Due to the competition, the performance of one users' partitioned execution not only depends on the partitioning itself, but also depends on the availability of the resources. Thus, we need to consider the profiling information from all the users, and make partitioning decisions that can guarantee the optimal average performance over the users rather than the optimal performance for each single user. Fig.1.1d shows that the partitioning is performed at the cloud side for all the users together with the scheduling function. The block 'P&S' in Fig.1.1d indicates the partitioning function coupled with scheduling.

1.3 Motivation of Our Work

Computation partitioning between the mobile device and cloud is an important and challenging research topic in mobile cloud computing. There is significant complexity involved in ensuring that the application can achieve adaptive and elastic computation partitioning between the mobile device and the cloud under dynamic environments and variable loads. Although some related issues are well solved, there are still many problems lacking sufficient investigation. In this section, we identify the research problems that need further investigation.

First, from the standpoint of end users, we need to optimize the trade-off between the local computation and data transmission. The mobile cloud application collects data from the mobile devices, and execute complex computations on the data. To reduce the resource consumption on the device, we can move some complex computations from the mobile devices onto the cloud. Although it speeds up the execution and reduces the computational cost on the mobile device as well, it incurs extra cost in data transmission. It is important to balance the trade-off between the reduced computational cost and the extra data transmission cost. We name this trade-off as computation offloading. More importantly, the optimal trade-off should be treated in practical applications together with the application functions development. Thus, we take the RFID tracking as one typical application, and apply the principle of the tradeoff into the RFID tracking algorithm design.

Second, if we consider complex applications, the trade-off between local computation and data transmission becomes more challenging. Complex application usually can be composed of a set of computations. The computations have data dependence between each other. The trade-off for each computation is dependent on each other. We need to achieve a global optimal trade-off for all the computations involved in the application. We name this global trade-off as computation partitioning. We find that few works study computation partitioning for data stream applications that are extremely popular in today's applications due to the proliferation of sensors attached to the mobile devices. Although some works have been done for the partitioning of streaming application, they focus on the

optimization of make-span rather than the data processing speed/throughput. Throughput is more significant in practical systems for the streaming application. It is urgent to develop new models and partitioning approaches for the data streaming application.

Third, if we consider the time property, the mobile cloud environment including the user's device status and network connection to the cloud can change with time during the course of application execution. We name it as dynamic mobile cloud environment. The partitioning of application should be updated adaptively depending on the changing of the dynamic mobile cloud environment. Most existing works assume that the users' environment remains static during the course of the application execution. They have one time partitioning when the application starts to run. The application sticks to the partitioning until the application ends. The one time partitioning approach yields significant performance degradation in dynamic mobile cloud environments. We need to develop online partitioning approaches under dynamic mobile cloud environments that can update the partitioning periodically during the course of application execution. To distinguish with previous works, we name it as *computation repartitioning*.

Fourth, from the standpoint of system, it is important for cloud providers to offer high performance partitioned execution to a large number of mobile users. The number of users can scale up and down very quickly. The users' partitioning should be able to adapt to variant number of input users and the provisioned cloud resources. Therefore, it is necessary to study the multiple user computation partitioning, where there exist a number of users who request for partitioned execution and compete for cloud resources. Existing works on computation partitioning study single user computation partitioning. These works assume that there exists no competition for resources at the cloud among the users and the cloud always has enough resources to execute the computations immediately when they are offloaded to the cloud. However, this assumption does not hold for large scale mobile cloud applications. In these applications, the offloaded computations may be executed with certain scheduling delay on the cloud due to the competition for cloud resources. Single user partitioning that does not take into account the scheduling delay on the cloud may

yield significant performance degradation. We need to study multiple user computation partitioning, which considers the partitioning of each user's computation jointly with the scheduling of computation onto the shared cloud resources.

In this thesis, we will analyze aforementioned problems in detail and propose corresponding solutions for them.

1.4 Contributions of the Thesis

The contributions of this thesis mainly lie in modeling, optimization and evaluation for computation partitioning in mobile cloud applications. As shown in Fig.1.2, in this thesis, we classify computation partitioning into different models by considering two dimensional properties: application dimension and system dimension. On application dimension, we do partitioning for two types of applications: computation dependent application and computation independent application. On system dimension, we do the partitioning for two types of systems: single user system and multiple users system. In this thesis, we focus on the study of three models: 1) computation independent application and single user system, named as *computation offloading*, 2) computation dependent application and single user system, named as *single user computation partitioning*, 3) computation dependent application and multiple user system, named as *multiple user computation partitioning*. The thesis contains three parts, which tackle the most urgent and significant issues in terms of the three models.

In the following, we will illustrate our contributions in detail one by one.

1.4.1 Contributions in Computation Offloading

Computation offloading is the simplest model of computation partitioning, where the application is composed of *independent* computations and the partitioning decision is done for one single user. In this part, we aim to optimize the trade-off between the local computation and data transmission. The trade-off should be integrated with the development of application, and validated in real world systems. Existing works on computation offloading focus on the models and algorithm design for one given application. However, in our

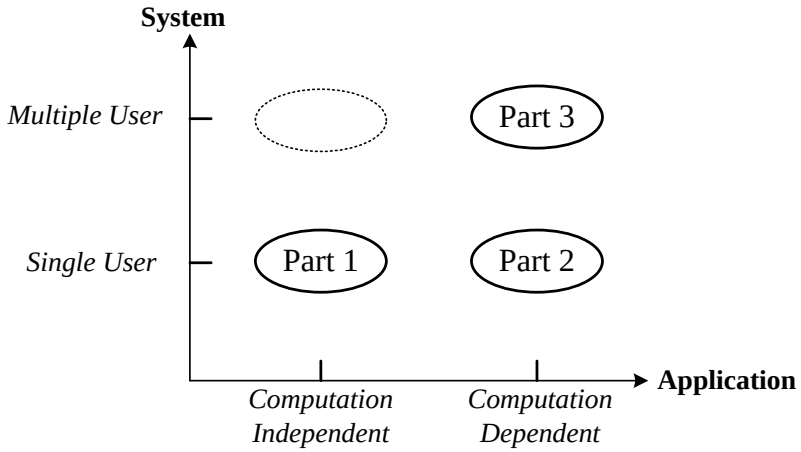


Fig. 1.2: The classification of models for computation partitioning

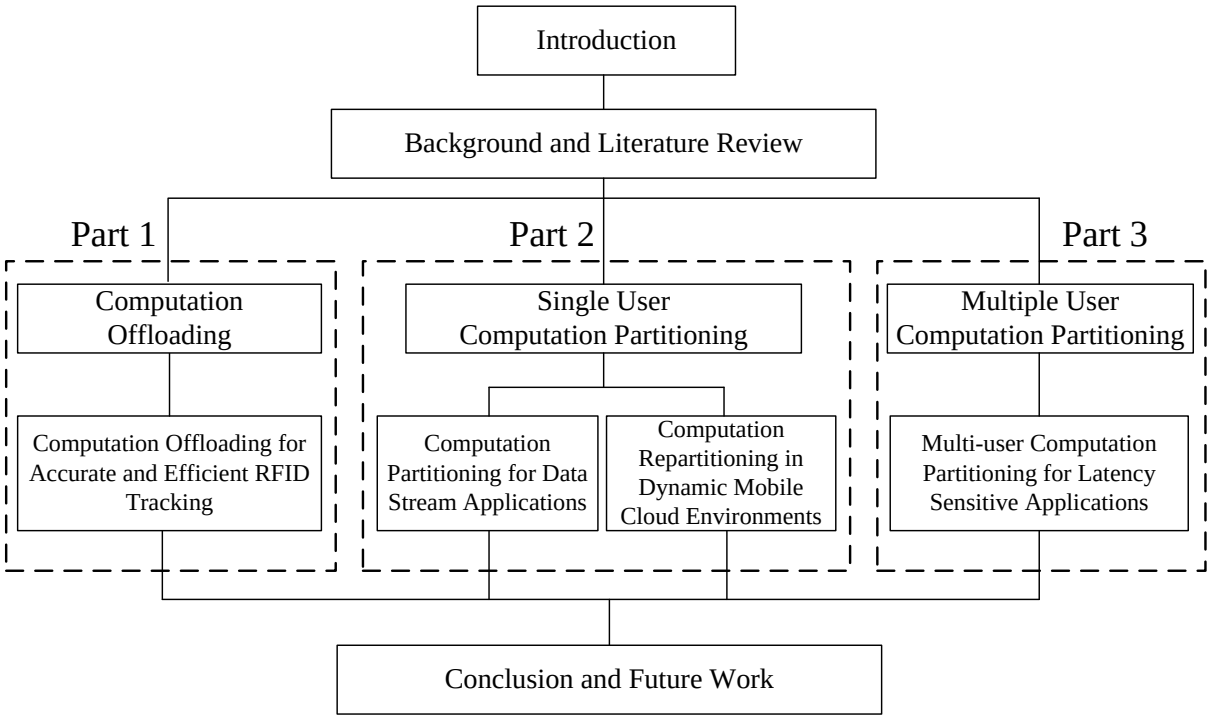


Fig. 1.3: An outline of the contributions of this thesis

work, we study how to apply the principle of computation offloading into the application development, and validate this principle through real world systems.

In particular, we take the RFID tracking as a case study. In the case study, we consider the RFID system that attaches the RFID reader on the moving object, and deploys passive

RFID tags in the environment. The moving object collects the noisy RFID readings, and perform continuous estimation of its position in real time. Traditional approaches such as Particle Filter (PF) can achieve high accuracy, but require a large amount of computations on the device. The approaches are hard to be implemented on some mobile devices that are constrained by the computing capabilities and battery. Other existing approaches such as Weighted Centroid Localization (WCL) are cheap in computational cost, but yield bad accuracy specially when the object's speed is high. Thus, we propose an adaptive approach to achieve accuracy and energy efficiency. Our approach can adaptively choose costly PF and cheap WCL according to the estimated speed of the object, and adaptively offload intensive computations required by PF onto nearby infrastructures or clouds based on the quality of network connections. We evaluate our solution through real world experiments, and show that our proposed approach with computation offloading outperforms other approaches in terms of both accuracy and efficiency.

1.4.2 Contributions in Single User Computation Partitioning

In single user computation partitioning, application is composed of dependent computations and the partitioning decision is done for one single user. We found that most existing works on computation partitioning pertain to this model. We study two problems that are important and not solved in existing works. The problems tackle the complexity involved in computation partitioning respectively in application dimension and system dimension. One problem is computation partitioning for data streaming application. The other problem is computation repartitioning for dynamic mobile cloud environment.

For the first problem, we aim at optimizing the partition of a data stream application between mobile and cloud such that the application has maximum speed/throughput in processing the streaming data. To the best of our knowledge, it is the first work to study 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 in other applications. We propose a framework to provide runtime support for the dynamic partitioning and execution of the application. The framework is

designed on the elastic cloud fabrics for better scalability. The optimization of the partitioning for each single user is by using genetic algorithm. Through both extensive simulations and experiments on real world applications, we show that our method can achieve more than 2X better performance over the execution without partitioning.

For the second problem, we study computation repartitioning under dynamic mobile cloud environments. Existing works with the one time partitioning of the application yield significant performance degradations in dynamic mobile cloud environments. Therefore, we consider application repartitioning problem which considers updating the partition periodically during the course of application execution. We first propose a framework for run time application repartitioning in dynamic mobile cloud environments. Based on this framework, we take the dynamic network connection to clouds as a case study, and design an online solution, Foreseer, to solve the mobile cloud application repartitioning problem. We evaluate our solution based on real world data traces that are collected in a campus WiFi hotspot testbed. The result shows that our method can achieve significantly shorter completion time over previous approaches.

1.4.3 Contributions in Multiple User Computation Partitioning

Multiple user computation partitioning is the most complex model of computation partitioning, where the application is composed of dependent computations, and the partitioning decision is made for multiple users. In this model, the multiple users compete for the computing resources shared by the users on the cloud. The users' partitioning decisions are dependent with each other.

We particularly study the Multiple user Computation Partitioning Problem (MCPP). We consider the partitioning of multiple userscomputations together with the scheduling of offloaded computations on the cloud resources. Instead of pursuing the minimum application completion time for every single user, we aim to achieve minimum average completion time for all the users, based on the number of provisioned resources on the cloud. We show that MCPP is different from and more difficult than the classical job scheduling problems.

In classical job scheduling problems, the computations are allowed to schedule onto arbitrary resources including the mobile devices and cloud resources, while in MCPP the users computations can not be scheduled to other users' devices. We design an offline heuristic algorithm, namely *SearchAdjust*, to solve MCPP. We demonstrate through benchmarks that *SearchAdjust* outperforms both the single user partitioning approaches and classical job scheduling approaches by 10% on average in terms of application delay. Based on *SearchAdjust*, we also design an online algorithm for MCPP that can be easily deployed in practical systems. We validate the effectiveness of our online algorithm using real world load traces.

1.5 Organization of the Thesis

The structure of this thesis is shown in Fig.1.3. Chapter 1 is the introduction to this thesis. Chapter 2 reviews related works in the literature. The main body of this thesis is divided into three parts from Chapter 3 to Chapter 6. The details are presented as follows.

In the first part, we present our work in computation offloading. In Chapter 3, we study how to use computation offloading into the development of applications. We take RFID tracking as an typical application, and design an accurate and efficient RFID tracking method using computation offloading.

In the second part, we present our work in single user computation partitioning. This part consists of two chapters. In Chapter 4, we study the computation partitioning for data stream applications. We propose a framework for partitioning and execution of data stream applications that includes the design of the architecture and algorithms. In Chapter 5, we study the computation repartitioning in dynamic mobile cloud environments. We design an online solution, *Foreseer*, for computation repartitioning that updates the partitioning periodically during the course of application execution.

In the third part, we present our work in multiple user computation partitioning. In Chapter 6, we study the multiple user computation partitioning for latency sensitive mobile cloud applications. We propose both offline and online algorithms to solve the problem, and evaluates the algorithms through benchmarks.

Finally, we conclude the thesis and discuss the directions of future works in Chapter 7.

Chapter 2

Literature Review

In this chapter, we present the literatures review on computation partitioning in mobile cloud applications. As introduced in Section 1.4, the contribution of this thesis contains three parts, computation offloading, single user computation partitioning, and multiple user computation partitioning. We discuss the related literatures in terms of the three parts from Section 2.1 to Section 2.3. In particular, most existing literatures on computation partitioning pertains to the single user computation partitioning. We spend most spaces to discuss the single user computation partitioning in details. In Section 2.4, we present the open issues on computation partitioning. The purpose of this chapter is to provide readers a broad view of the state of the arts and open issues that the researchers are focusing on in this area.

2.1 Computation Offloading

The mobile cloud system contains three parts, mobile devices, wireless access and clouds. The mobile devices can offload some computations of the application to the clouds. Obviously, offloading can reduce the computational cost (e.g., execution time or energy consumption) of the mobile device. Meanwhile, offloading causes additional overhead in data transmissions that are required by the remote execution on clouds. If we treat the application as a black box which has computational cost if executed locally, and data transmission cost if executed remotely, we can decide whether the application should be executed locally or remotely. However, this level of offloading decision for the application is too coarse.

For some complex applications which can be divided into a set of parts, we need to make offloading decisions for every part of the application.

Now we are interested in the question: what affects the offloading decisions. First, the offloading decision depends on the device information such as the execution speed of the device, and the workloads on the device when the application is running. For example, if the device computes very slowly and the aim is to reduce the execution time, it is better to offload the computations onto clouds. Second, the offloading decision depends on the network bandwidths. This factor affects the cost in data transmission for remote execution. For instance, if the bandwidth is high, the cost on data transmission will be low. In this case, it is better to offload the computation onto clouds. Third, the partitioning decision also depends on the application itself. The application requires computations and data transmission if offloaded to the cloud. We name the ratio between the amount of computations (in term of execution instructions) and data transmitted as Compute to Communication Ratio (CCR). If CCR is high, this application is compute intensive, and is better to be executed remotely; otherwise the CCR is low, the application is data intensive, and may be inherently suitable to be executed locally. Unlike the device and network information, CCR is a static factor that influences the offloading decision. Different applications normally have various CCR values. The device information and network information usually change with time. The collection and estimation of device and network parameters in real time is called *profiling*. Profiling is a challenging issue, and will be discussed in next section. With profiling information about the device and network parameters, we will compare the computational cost of local execution and data transmission cost caused by remote execution, and choose the less costly one.

Computation offloading is the most widely used technique to solve the resource poverty problem of mobile devices in mobile cloud computing [WGKN08] [YOC08]. Karthik et. al [KL08] argues that cloud computing could potentially save energy for mobile user, but not all applications are energy efficient when migrated to the cloud. It depends on whether the computation saved due to offloading outperforms the communication cost. According

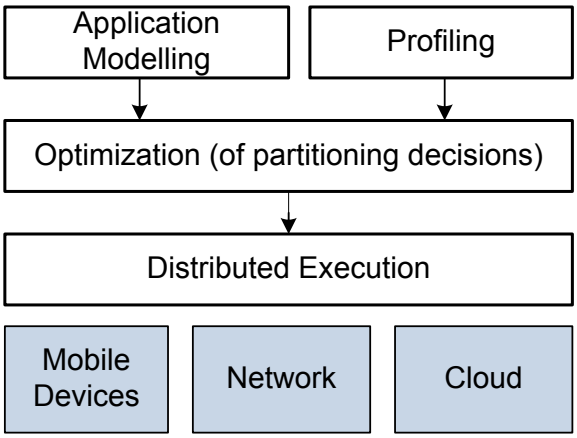


Fig. 2.1: General Components of a Computation Partitioning System

to the cost model, the offloading will bring benefit to energy saving if the application has a large computation-to-communication ratio and runs in a networking environment with good connectivity. M. Satyanarayanan [SBCD09] presents a computing model that enables a mobile user to exploit VMs to rapidly instantiate customized service software on a nearby cloudlet and uses the service over WLAN. A cloudlet is a trusted, resource rich computer or a cluster of computers well connected to the Internet and available for use by nearby mobile devices. Rather relying on a distant cloud, the cloudlets eliminate the long latency introduced by wide-area networks for accessing the cloud resources. As a result, the responsiveness and interactivity on the device are increased by low-latency, one-hop, high bandwidth wireless access to the cloudlet.

2.2 Single User Computation Partitioning

In this section, we introduce the literatures in terms of four general components in single user computation partitioning: application modeling, profiling, optimization and distributed execution. Application modeling aims to represent the dependence relationship among the divided computational modules in the application. Partitioning of the application depends on a few factors such as the device capability and workload, network bandwidth, and the application itself. The device information and network bandwidth are dynamic, and can vary with time in practical systems. Profiling is to collect and estimate the dynamic

Table 2.1: Literature review on computation partitioning

	App. Modeling			Profiling		Optimization		Distributed Exec.		
	Proc. Call Model	Service Invoc. Model	Data Flow Model	Predi. based Profiling	Model based Profiling	Online	Offline	Client Server Comm.	VM Migration	Mobile Agent
MAUI [CBC10]	√				√	√			√	
CloneCloud [CIM+11]	√				√		√		√	
Odessay [RSM+11]			√	√		√		√		
Giurgiu et.al. [GRJ+09]		√			√	√		√		
Zhang et.al [ZKJG09]		√			√		√	√		
Yang et.al [YCY+13]			√		√	√		√		
Spectra [FPS02]		√			√	√		√		
Chroma [BSPO03]		√			√	√		√		
Scavenger [Kri09]		√			√	√				√

information of device and network connections to the cloud. Given the profiling information, the core part of the system is to optimize the partitioning of the application, such that the execution cost for each single user is minimized. According to the partitioning results, the system will launch an distributed execution of the application between the mobile device and the cloud. Fig.2.1 illustrates the three components a computation partitioning system. Table 2.1 summarizes the literatures on the four components. In the following, we will describe the details in term of the four components

2.2.1 Application Modeling

Application model refers to two meanings: 1) the programming model according to which the programmers develop the application; 2) the mathematical model that represents the structure of the application. The former one is to provide programming abstractions for the application development, while the latter one provides the formal representation of the application that is to be partitioned. The latter one usually depends on the programming model. Thus, we describe the application model from the perspective of programming. According to our survey, there exist three application models: procedure call model, service invocation model, and dataflow model. In the following, we discuss the models and related literatures.

Procedure call model. In [CBC10] [CIM+11] [KAH+12], the application is represented by a set of procedures. Each procedure can call other ones. Thus, we can use a procedure call tree or graph to model the structure of the application. In the tree/graph, the node represents the procedure, and the edge represents the call relationship. The programmers write the application by the principle of procedure oriented paradigm. The partitioning problem is to decide for each procedure whether it should be offloaded or not. Procedure call model is used for the procedure/method level partitioning of application. The partitioning of application is fine-grained. The model can be applied to represent most applications.

Service invocation model. Under service invocation model, the application is composed of a set of services. We usually use a service invocation graph to model the application, in which the node indicates the service, and the edge indicates the service The programmers need to program the application using the service oriented methodology. [ZKJG09] and [GRJ+09] pertain to the application model. [ZKJG09] decomposes the application with of a set of 'weblets'. The weblet is actually a kind of web service, and can be executed at either the mobile side or the cloud side. [GRJ+09] build their partitioning system based on a distributed service computing platform, named as AlfredO [RRA08], which have been traditionally used to decompose and couple Java application into software modules.

Note the service invocation model and procedure call model decompose the application at different granularity. Service invocation model supports the module-level partitioning of application, and decomposes the application according to the functional modules. The decomposed modules are loosely coupled. The procedure call model decomposes the application according to the structure of the code. The decomposed procedures are tightly coupled with each other, which brings programming difficulties in distributed execution. , Compared with procedure call model, service invocation model is more coarse-grained. However, from the perspective of representation methodology for the application, service invocation model and procedure call model are the same. They use very similar graph to model the application.

Dataflow model. Dataflow is suitable to model most media applications that have continuous in-coming data to process. In this model, the application is composed of a set of dependable stages. Data flows between the stages. The output data of the stage becomes the input data of the next stage. Each stage performs one particular function onto the in-coming data. Dataflow can be represented with a directed acyclic graph in which each node is the stage, and each edge indicates the data dependence between the two connecting stages. In [RSM+11] and [YCY+13], the application to be partitioned is modeled as dataflow graphs. Fig.3 shows the dataflow graph of the face recognition application.

2.2.2 Profiling

The profiling is the estimation of the execution cost of each part of the application. There exists two profiling approaches: *prediction based profiling* and *model based profiling*. Prediction based profiling is estimating the execution cost from the records of execution cost in past execution instances. Execution instance is defined as one-time execution of the application. In prediction based profile, we record the execution cost in every execution instance, and predict the current execution cost from the historical records. Model based profiling is estimating the execution cost based on a model or function, which takes the application information such as the workload of the computations and size of data transmission between two dependent computations, device status and network status as input,

and the execution costs as output. In model based profiling, we do not directly measure the execution cost. Instead we measure the device and network status, and calculate the execution cost from these measurements.

Odessay [RSM+11] uses the prediction based profiling approach. The execution time of each computation is updated once the application is executed again. The execution time in the latest execution instance is used as the cost to determine the optimal partition. The most recent works on computation partitioning [ZKJG09] [KAH+12] have the same profiling approach with Odessay. Prediction based profiling approach avoids the overhead of real time measurement of network and device status. The accuracy depends on the freshness of the latest execution instance.

MAUI [CBC10] uses the model based profiling approach. MAUI [CBC10] estimates the energy consumption of each part of the application based on a model that represents the energy consumption as a function of the CPU cycles. The model is learned offline through real measurements. The authors of MAUI evaluate the accuracy of the model, and show that the error produced by the model is less than 6%. MAUI [CBC10] estimates the remote execution time of each part of the application through an online measurement of the network bandwidth. It estimates the bandwidth by observing the size of data transmission in recent offloading and the time spent in the transmission. If there is no offloading recently, the system transfers one 10K file to the server to test the network bandwidth. CloneCloud [CIM+11] also pertains to model based profiling approach. However, the paper does not discuss how to estimate device status and network bandwidth in real time. Model based profiling approach requires the online estimation of the device and network status. This online estimation may cause additional overhead on mobile devices. However, it has better accuracy than the prediction based profiling approach, especially in the dynamic mobile cloud environment.

2.2.3 Optimization

The computation partitioning requires the optimization of partitions based on the profiling execution cost of each part of the application. The objective is to determine an optimal

partition to minimize the total execution cost. The execution cost can be in terms of execution time, data processing throughput, energy consumption. MAUI [CBC10] aims to optimize the energy consumption on devices. CloneCloud [CIM+11] and ThinkAir [KAH+12] support the optimization of either the execution time or the energy consumption, which depends on the programmers' choices. Odessay [RSM+11] aims to optimize the makespan for data streaming application, while [YCY+13] first propose to optimize the processing speed/throught for streaming application. [ZKJG09] has a hybrid optimization objective that can be customized by the end users.

According the time that the optimization is performed, we classify existing literatures into two categories: offline optimization and online optimization. Offline optimization determines the optimal partitions for various execution conditions at offline phase. Execution condition includes the device computing capability and network bandwidth. The execution conditions and corresponding optimal partitions are stored in the database. During online phase, one of the partitions is selected from the database according to the estimation of current execution condition. Online optimization solves the optimization on the fly according to the profiling cost. The online approach has good accuracy, but it brings overhead caused by the optimization. [CIM+11] and [ZKJG09] adopts the offline approach, while [CBC10] [RSM+11] [YCY+13] use the online approach.

For the online approach, the place of optimization can be at the mobile side, or the cloud side. Most existing works [CBC10] [CIM+11] [BKMS13] [RSM+11] place the optimization on the mobile device, while [YCY+13] supports the optimization on the cloud side. The optimization on the mobile device does not require the profiled parameters transmission over network. It causes additional compute overhead on the device. The optimization at the cloud side can avoid the compute overhead on the device, but requires continuous connection with the mobile device to transmit the profiled parameters. It is suitable for the partitioning of complex applications.

2.2.4 Distributed Execution

We classify the approaches of distributed execution into three categories: client-server communication, VMs migration, and mobile agent. In client-server communication method requires the pre-installation of the program codes on the cloud servers. When one function of the application is decided to offload onto the cloud, this function is usually performed through the protocol of Remote Procedure Call (RPC) or Remote Method In-vocation (RMI). [FPS02] and [BSPO03] use the client-server communication method to implement the partitioned execution. The method's disadvantages are that it is prone to fail under network disconnection. The codes on the cloud/server side need to be changed from the original codes on the mobile device. The deployment of the partitioning system is not convenient.

[CBC10] [CIM+11] [SBCD09] [KAH+12] propose to implement the partitioned execution by Virtual Machine migration. At the mobile side, the application is running on a virtual machine. When some part of the application is decided to offload onto the cloud, the whole virtual machine would migrate to the cloud side. The virtual machine would migrate back to the mobile side, when the part of application is finished on the cloud. The method does not require pre-installation of application on the cloud side. However, the VM migration could incur more overhead than remote procedure call method, due to the transmission of execution state of the virtual machine such as the memory and registers state.

Scavenger [Kri09] is the early work that uses the approach of mobile agent to implement the remote execution. It provides a platform that can help users easily programming and de-ploy partitioning enabled applications. Dynamic deployment of application is realized in this approach. However, it needs agent management that causes overhead on the mobile devices.

2.3 Multiple User Computation Partitioning

Most existing works pertain to single user computation model [CBC10] [CIM+11] [ZKJG09] [GRJ+09] [LWX09] [KAH+12] [RSM+11] [BSPO03] [YCY+13]. A critical assumption behind this model is that the resources shared by the users are always enough, such that the allocation of the resources does not influence the execution of application that has been partitioned in advance. For instance, it is assumed that the cloud always has enough servers to accommodate the computations offloaded from the mobile device. Suppose the metric to be optimize is execution time. The offloaded computations should be executed on the servers without any delay; otherwise, the performance of the partitioned execution will be sacrificed. This assumption holds in the cases where the clouds have nearly unlimited computing resources, or the number of mobile users that can offload computation onto clouds do not exceed to the clouds' capacity. The single user model is suitable to apply into the system that serves for small or predictable number of users. The cloud always guarantees optimal partition for each single user.

Multiple user computation partitioning model is first proposed in our previous work [YCCJ13]. The multiple user model applies into the scenario that the workloads/computations from mobile users exceed the capacity of the cloud, such that the users need to compete for the resources at the cloud side. In this scenario, instead of achieving optimal performance for each single user in multiple user model, the objective is to achieve the optimum of the overall performance. We argue that single user model is suitable to be used in large scale system with unpredictable workloads. The coupling between partitioning and allocation of shared resources makes the partitioning problem under single user model more challenging.

2.4 Challenging Issues in Computation Partitioning

2.4.1 Energy Efficiency

As the mobile device has increasing processing capability, the energy consumption becomes the major issue for mobile applications. Most device vendors look for approaches to increase the battery life. Besides inventing new battery technologies, there are a lot of

methods to save the energy consumption at the system and application layer. Computation partitioning is considered as an important approach to save energy consumption on devices. By using the approach, the components of the application that consumes a lot energy, e.g., compute-intensive algorithms, are offloaded onto clouds. However, the difficulty in this approach is to design effective mechanisms to monitor and profile the energy consumption for the applications on mobile devices. Designing the models for the estimation of energy consumption in data transmission is not easy as well. Both the profiled information and models are critical to partitioning the application for energy saving.

We need to design the light-weight and energy efficient supporting softwares for the partitioning of application. The costly function in a computation partitioning system is optimization. The optimization can be performed offline. The partitions that are generated from the offline optimization are stored on the mobile devices. Whenever the execution environment changes, the application will be configured with the optimal partition from all the back-up partitions. The offline optimization can save the energy overhead. [**?**] is the early work that proposes to implement the offline optimization. Another implementation method for energy saving is to perform the optimization on the cloud. [YCY+13] proposes a partitioning framework that implements the optimization on clouds by using the genetic algorithm.

Other researchers look for techniques to optimize the energy consumption in data transmission. Offloading computations onto clouds require transmissions of the input data of the computations. The issues on energy consumption in data transmission need to be tackled in computation partitioning. E. Uysal Biyikoglu et.al [UBG04] design an energy efficient data transmission mechanism. The mechanism monitors the network quality, and transmit the data based on network quality. If the network quality is good, data is transmitted; otherwise, no data is transmitted for saving energy.

2.4.2 Mobile Access Management

In the mobile cloud partitioning system, the mobile access networks such as 3G/4G cellular networks and Wireless Local Area Networks (WLANs) are important part to connect

the mobile devices and the cloud. The quality or bandwidth of the user's connection to clouds will directly determine the partitioning of the application.

Network intermittence

One practical issue is how to partition the application under intermittent network connections. As described in last section, the application is usually partitioned based on a cost model that contains computational cost respectively on mobile side and cloud side, and offloading cost, e.g. data transmission cost. Most works assume that the offloading cost does not change during the run time of application. This is not practical in mobile environment. In reality, the network connectivity can fail due to wireless network holes, which is defined as places where there is no signal or signal is too weak to maintain a connection. Even when the network is connected, the throughput or bandwidth can fluctuate because of user's mobility. The fluctuant network status leads to varying offloading cost.

C. Shi al.et. [SAZN12] consider the intermittent connectivity of the network, they solve the problem by assuming the future network connectivity is perfectly known. They designed an offline algorithm to calculate the optimal partition given the future network bandwidth. In practical systems, we need to design online algorithm to partition the application [YCT+13]. The algorithm can update the partition of application from time to time during the course of the application execution, according to the prediction of future network status. The prediction of network bandwidth is also a critical issue we need to tackle.

Several previous works predict future network status based historical mobility observations. The mobility prediction based approaches have been used in other applications and systems. The first application domain is in WSN data delivery. [KLW+09] achieve gains in routing performance using a mobility prediction algorithm. [LWK+10] study the problem of delivery data from data source nodes to the mobile sink. It predicts the nodes that the mobile sink is likely to pass by, and then stashing data on these in advance. There exists as well a lot of early works on mobility prediction in cellular/Wifi networks that aim to improve the network hand-off performance by predicting the next cells/APs [LBC98] [LBC06].

Network resource allocation

Another issue is network resource/bandwidths allocation for multiple user computation partitioning. Consider that the mobile users offload computations onto clouds through the same access networks. The resources or bandwidths of the networks are limited. We need to allocate the resources to the mobile users. The user who is allocated with more bandwidth has less offloading cost, while the user allocated with less bandwidth will has high cost in computation offloading. The users' partitioning decisions are dependent with each other because of their competition for the shared network resources. Thus, the partitioning problem needs to be solved jointly with the network resource allocation.

The network resources may contain both the cellular networks and WLANs. The research problem can be in different ways. One ways is that, given the cellular network and WLANs resources, we need to allocate the cellular and WLAN networks resources to mobile users, such that the overall system performance is maximized. Another way is that, given the mobile users, we need to determine how much each type of network resources should be leased by the application provider, and how to allocate the resources to mobile users, such that the performance with lowest cost is achieved.

2.4.3 Workload Management

Workload management is another important issue in computation partitioning. The work-load refers to the computations offloaded from mobile users onto clouds. The mobile cloud application needs to have the ability to server a large number of mobile users. When the application scale increases, how to manage the offloaded workload from mobile users at the cloud clusters is essential to the cloud resource usage efficiency and system performance. In single user computation partitioning model, the workload management issue is unrelated to the computation partition of each user. Traditional workload scheduling and balancing mechanisms [HGSZ10] [FWG10] [MSY12] can be used to tackle the problem. Next we discuss the workload management issue in multiple user computation partitioning.

Workload scheduling in centralized cloud

In multiple user computation partitioning, the workload management is correlated with the application partition of each user. To achieve good system performance, it is better to consider the computation partitioning and cloud workload management together. We first considers the problem under a simple system model [27]. The application is modeled as a sequence of dependent tasks. The mobile users run the same application. On the centralized cloud there exists a set of server nodes that accommodate the workload (tasks) offloaded from the users. The objective is to schedule the tasks for all the users onto their mobile devices and the cloud servers. Each user's device can only execute the task from itself, and it can not execute tasks from other users. The problem is abstracted as a job scheduling problem which is similar to but more difficult than traditional job scheduling problems in parallel computing.

Workload scheduling in distributed clouds

The workload scheduling in multiple user computation partitioning becomes more challenging when we consider that the cloud consists of geographically distributed data centers. Under this model, there exists a set of mobile users from different regions. Each user has a partitioned execution of the application. The cloud contains a set of data centers that are distributed in different regions. Each data center has certain capacity in term of computing resources. For each user, offloading the same component onto various data centers can lead to different offloading cost, since the network connection to data centers has different delay and/or bandwidth. We need to partition the application for the mobile users, and as well allocate the offloaded computation onto the computing resources at different data centers. We have two types of workload scheduling, i.e, inter data center scheduling, and intra data center scheduling. Both the two scheduling should be considered jointly with the partitioning of application for each user, such that the overall system performance, e.g., the application execution time, is maximized.

Several recent works [GCWK12] [WLZ$^+$12] [WLC12] in cloud computing have been done

to solve the scheduling problem for distributed clouds. P. Gao et.al [GCWK12] develops an optimization framework to schedule the data accessing requests/workloads from users onto distributed data centers. The scheduling problem is studied with the aim to minimize the energy consumption on the clouds. Y. Wu et.al [WLZ+12] studies the scheduling of Video-on-Demand accessing re-quests/workloads onto the geographically distributed clouds. The scheduling problem is studied together with the video placement problem. The objective is to minimize the operational cost while satisfying the delay constraints for the video accessing requests. [WLC12] also studies the requests scheduling problem for live video streaming application. However the existing works [GCWK12] [WLZ+12] [WLC12] can not be applied into the workload scheduling in computation partitioning systems, because the scheduling and partitioning are coupled in multiple user computation partitioning and can not be treated separately.

2.4.4 Performance Modeling and Monitoring

An important issue we face is how to design the performance model for various mobile cloud applications. We consider three different types of mobile cloud applications, content delivery centric application, sensing delivery centric application and user interactive application. These types of applications have different performance requirements from the perspective of both end-users and systems. We need to design an accurate performance model which can illustrate both the end-user and system related performance requirements.

To solve the issue involved in performance model development, we have investigated a lot of works and ISO standards [Raj91] in the area of software engineering that proposed appropriate performance models for various software systems. In Jain's model [Raj91], the system is supposed to have three outcomes for a given request. It may perform the request correctly or incorrectly, or it may refuse to perform the request. Three concepts on the performance have been correspondingly defined: 1) speed, 2) reliability, and 3) availability. It is needed to develop the performance model of mobile-cloud partitioning system by adding more practical factors such as SLA, time behavior of the system, utilization, capacity and recoverability and so on.

Another issue we face is how to detect the anomalies or performance degradations for mo-bile-cloud partitioning system. In traditional Internet applications, anomaly detection is done by manually analyzing the logs [NKSH09]. This method is not feasible for large-scale cloud system. Some researchers apply pattern recognition approach [KM09] [LB99] to automate the analysis of massive volumes of system logs. Since complex computational cost in the analysis, this approach is not feasible for the system that requires real time anomalies detection and recovery. Besides, the logs analysis based methods are not enough to detect the anomalies or performance degradations for mobile cloud application. Usually, the performance of mobile cloud application depends on more complex factors which could include the failures or inefficiency at mobile devices, wireless networks as well as clouds. Collecting logs from mobile devices for analysis may not be possible due to high cost or privacy issues. Faced with these difficulties, we need to develop suitable anomly/performance degradation detection approaches for mobile cloud applications.

To solve the anomly/performance anomaly detection problem, one possible way is to use a hybrid method that integrates the logs analyzing technique and real time performance monitoring technique. At the cloud side, we detect anomalies by analyzing the system logs, while at the mobile side we will design lightweight performance monitors. We need to design network protocols for the interactions between mobile and cloud for the performance monitoring.

Chapter 3

Computation Offloading for Accurate and Efficient RFID Tracking

In this chapter, we study how to apply computation offloading into the design of accurate and efficient RFID tracking algorithm. This chapter is organized as follows. We present an overview of this work in Section 3.1, and the background on RFID localization and tracking in Section 3.2. We describe the system model and RFID sensing model in Section 3.3. Section 3.4 presents the hybrid method for object tracking. The computation offloading for RFID tracking application is described in Section 3.4.5. The simulation results is presented in Section 3.5. In Section 3.6, we validate the performance results of our method by implementing it in two real systems. We conclude this chapter in Section 3.7.

3.1 Overview

As one of the most challenging problems in the area of mobile service, localization has been extensively studied over the past decade. Compared with existing solutions using laser scanners, cameras, or ultrasound, Radio Frequency Identification (RFID) technique has attracted a lot of interest in recent years due to its wide adoption by the industry. According to the types of the used tags, the techniques are classified into two categories,

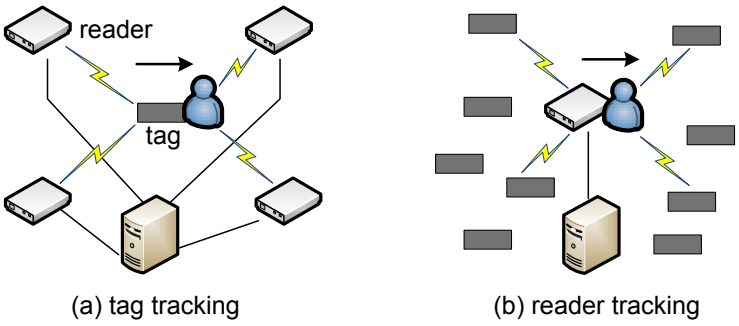


Fig. 3.1: RFID tracking

active tags based technique [NLLP03][HJG06][HJG07][YTH04] and passive tags based technique [HBF$^+$04][JPB09][GM10][HLL07][PH09][SV07][VZ08][LCS06]. Passive tags are more attractive than active tags because of its low cost and convenience for large-scale deployments [BT08].

In this chapter we study the problem of location tracking using passive RFID tags. For convenience, the tags refer to the passive RFID tags throughout this chapter. RFID based location tracking can be classified into *tag tracking* and *reader tracking* [NLLP03]. In tag tracking as illustrated in Fig.3.1(a), the object to be located is attached with a tag. The RFID readers are deployed in the environment. As the object moves in the environment, the readers collect the data. The readers can either deliver the data to a centralized server that calculates the position, or cooperate with each other to estimate the position by themselves. The positioning result is then returned to the object. In reader tracking as illustrated in Fig.3.1(b), each object to be located carries an RFID reader as well as an antenna integrated with the reader. The RFID tags are deployed in the environment. The reader obtains the data and calculate its own position. Compared with tag tracking, reader tracking reduces infrastructure cost by using cheap tags instead of expensive readers. We focus on reader tracking in this chapter.

The RFID reader tracking has three challenges. First, we focus on the tracking of

mobile objects rather than locating of stationary objects. It is more challenging because the algorithm to estimate the object's current location must be executed before a deadline in order to meet the accuracy requirement. Second, the RFID readings gathered from real world are noisy. It means that each tag in the reading range of the RFID reader is not certainly but possibly to be detected. Detection failure of tags is normal. This assumption is demonstrated in Section 3.3 by the RFID sensing model measured from experimental data. Third, in the RFID reader based tracking system, the continuous and real time estimation of object's location on the embedded RFID device requires very computation efficient tracking algorithm.

Existing methods for tracking mobile objects using noisy RFID readings are often based on using Kalman filter, and Particle Filter (PF) [HBF+04][GM10][SV07][VZ08], also known as Sequential Monte Carlo Method [TFBD00]. Kalman filter is suitable to be used in linear system with Gaussian noise. However, in many practical RFID tracking systems, the location evolves none-linearly over time, and the noise may not be Gaussian. In these none-linear none Gaussian systems, particle filter can get more accurate solution than Kalman filter. In particle filter, the object's position is represented by a set of weighted samples (called particles). Two procedures, sampling and filtering, are executed repeatedly. During the sampling, the particles are sampled randomly from an area which is usually predicted according to the motion model of the mobile object. During the filtering, each particle is first assigned a weight according to the sensing model. The particles with low weight are then filtered out. However, continuous execution of particle filter suffers from high computational cost on a resource constrained RFID enabled device. Other existing approaches [BT08] [BGGT07] [BHE10] such as Weighted Centroid Localization (WCL) are cheap in computational cost, but yield bad accuracy specially when the object's speed is high.

In this chapter, we propose a hybrid method for tracking mobile objects with high

accuracy and low computational cost. This method particularly uses two critical techniques. First, it adaptively chooses costly PF and cheap WCL according to the estimated speed of the object. If the mobile object's velocity is less than a threshold, WCL is used; otherwise, the particle filter is used. This is because of our observation that, when the object's speed is low, WCL is able to achieve at least the same accuracy with particle filter while costing much less computation. Oppositely, when the object's speed is high, the accuracy of WCL is much worse than particle filter. Furthermore, the motion pattern of the moving object (e.g. orientation) becomes stable under high velocity. This stable pattern could be taken advantage of to improve the sampling efficiency, if particle filter is used, and thus to reduce the computational cost. The complementary property of PF and WCL in term of accuracy and efficiency motivates us to integrate them together.

Second, our hybrid method can adaptively offload intensive computations required by PF onto nearby infrastructures or clouds based on the quality of network connections. We evaluate our solution through real world experiments, and show that our proposed approach with computation offloading outperforms other approaches in terms of both accuracy and efficiency. We summarize the contribution of this chapter as follows:

- We propose a hybrid method for object tracking using noisy passive RFID readings. Compared with existing particle filter method, it saves a lot of computational cost while achieving the same accuracy.

- We design a fast and cheap method to estimate mobile object's velocity from the RFID readings. It is not necessarily applied in our proposed hybrid method for object tracking, but can be used separately in other application scenarios which require the measurement object's speed.

- Besides extensive simulations, we have validated the method in two real systems, indoor wheelchair tracking and LRVs (Light Rail Vehicles) tracking in one of Hong Kong MTR depots.

3.2 Background

There is a variety of approaches to RFID-based localization, which can be characterized by the type of devices that have been used. One is RFID localization based on active tags. Most of the early systems use active tags due to its similarity in principle with other RF localization approaches, such as WLAN and WSN localization. Lionel et al. [NLLP03] present LANDMARC that uses kNN technique for localizing unknown active tags. Reference tags with known positions are deployed regularly on the covered area. The approach consists of selecting the k nearest reference tags from the unknown tag by comparing the signal strength of tags at unknown location with signal strengths received at reference node. Similarly by deploying reference active tags and readers in environment, Huang et al. [HJG06] utilize Bayesian inference to calculate the target position. The RF propagation parameters are firstly calibrated using on-site reference tags, and then the distance between the targeted tag and the readers is estimated with a probabilistic RSS model. Wang et al [HJG07] applies a different method to estimate the distance between the reader and target tag. For localizing a tag, the readers start with lower power level and gradually increase the transmission power until they receive the response from the tag. The distance between a reader and a tag is then estimated by averaging the distances from the reader to all reference tags detected in the same power level. The simplex method is finally used to compute the target location given its distances with readers. Yamano et al. [YTH04] demonstrates how support vector machine could be used to learn robot location. Feature vectors are generated out of signal strength information gained from active RFID

tags.

Another type of RFID localization approach is based on passive tags. This approach becomes widely used for target localization due to its cheapness. However, unlike the active tags, passive tags do not provide any information about signal strength. Also, the communication between the passive tags and the reader is more sensitive to environment settings such as the tagged materials. Therefore, localization using passive RFID tags is generally more challenging. Han et al. [HLL07] present localization scheme for an indoor mobile robot using passive RFID tags. A triangular pattern of arranging the RFID tags on the floor is adopted to reduce the estimation error of the conventional square pattern. Park et al. [PH09] propose a method using read-time of IC tags to reduce the localization error of an RFID navigation system. Both of these two literatures assume a binary sensor model and do not consider detection failure of passive tags.

Hahnel et al. [HBF+04] first obtained a probabilistic sensor model for their RFID reader which indicates the likelihood of detecting an RFID tag given its relative position to the antenna. This model is used to localize the tags and then to localize a mobile antenna given the tag map. Many literatures in localization with passive tags follow in this way [HBF+04][GM10][SV07][LCS06]. Vorst et al. [HBF+04] designed a method to learn a tag detection sensor model in a semi-autonomous fashion. Joho et al. [JPB09] proposed probabilistic sensor model that characterizes the received signal strength as well as the tag detection in order to achieve a higher modeling accuracy. However, in these literatures, the RFID reader used to localize object has a large reading range, as long as several meters, making it difficult to obtain a close-grained sensor model. The positioning accuracy is limited by this coarse-grained sensor model.

3.3 Models

3.3.1 System Model

The RFID reader based tracking system contains three components: RFID reader, a number of passive RFID tags, and the environment infrastructures, which are shown in Fig.3.1(b). The reader is attached on the object to be tracked. The infrastructures include the servers deployed in the area where the object is moving. The RFID reader can communicate with the server through wireless networks such as WLAN. We assume that the RFID reader is integrated with a circularly polarized antenna. Fig.3.2 illustrates the geometrical relationship of the reader and tags. The reader has a height of H above the ground, and a circular reading range tr. The number of passive tags deployed in the tracking area is K. Each tag's location, denoted as T_i, is stored in advance at the reader in association with its tag ID. Whenever the reader detects a tag, it can access the tag's location through the tag ID. The target moves with a velocity of v. The RFID readings are obtained in a frequency of f.

It is assumed that detection failure could happen even when the tag appears in the reader's reading range. (Actually this failure always happens in real word because of either the hardware failure or wireless environment inference). Let RFID reading at time k be $z_k = \{r_k^i\}_{i=1}^K$, where r_k^i indicates whether or not tag i has been detected at the time k; if it has been detected, then $r_k^i = 1$; otherwise $r_k^i = 0$. The objective is to estimate the continuous location of the mobile object using the uncertain RFID reading. The following section describes a RFID sensing model in order to capture this uncertainty of RFID reading.

3.3.2 RFID Sensing Model

Previous works [HLL07][PH09] in RFID localization assume that a perfect binary sensor model exists. The reader is capable of detecting the tags that lie within its sensing range

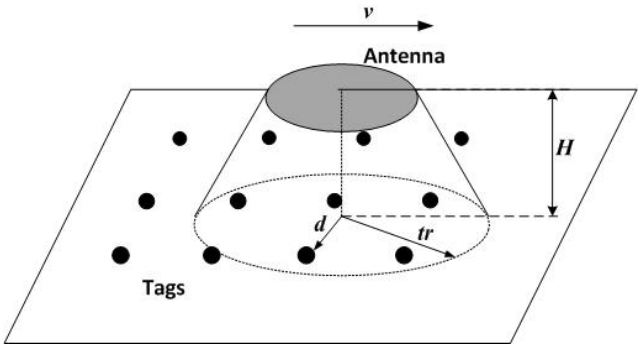


Fig. 3.2: System model

Table 3.1: Summary of terms and their definitions

Term	Definition
$p(r)$	RFID sensor model
tr	Reading range of RFID reader
f	Observation frequency of the RFID reader
K	The number of tags
T_i	The location of i-th tag
v_{th}	Threshold of the velocity
N_{th}	Threshold of the maximum duration
r_k^i	Detection status of tag i at time k.If detect, $r_k^i = 0$; otherwise,$r_k^i = 1$
z_k	The RFID reading at time k, $z_k = \{r_k^i\}_{i=1}^K$
l_k	Location of target at time k
β_i	The spatial weight of tag i
$\lambda_t(k)$	Temporal weight in Extended WCL. k is historical time, t is current time

and not the tags beyond it. In reality, the successful detection of a specific tag depends on a large number of parameters: 1) relative position of a tag from the reader; 2) absorbing and reflecting materials in the environment; 3) noise existing in the environment. These parameters lead to uncertainty of successful detection of tags by the reader. To ensure that our algorithm is as true to reality as possible, we assume a probabilistic sensing model by Equation (3.1), which is motivated by the existing sensor model proposed in WSN

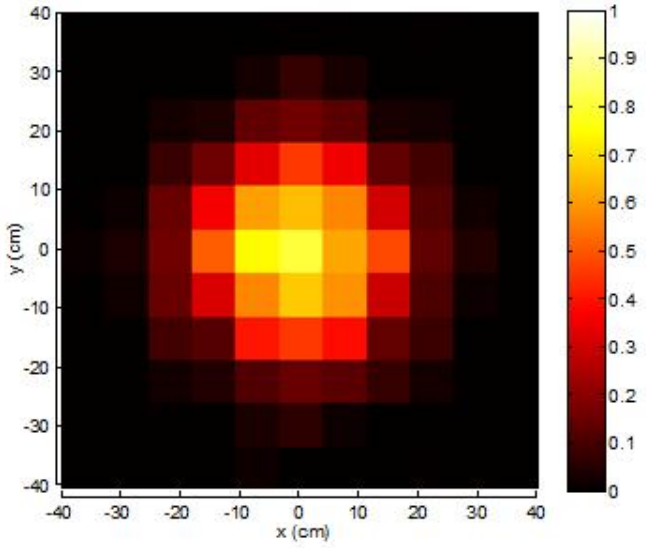


Fig. 3.3: Detection count and sensor model measured by experiments.

[BHE08][ZC04].

$$p(r) = \begin{cases} \lambda e^{-\beta r^{\alpha}}, r < tr \\ 0, r > tr \end{cases} \tag{3.1}$$

All tags that lie beyond the distance tr have a probability of 0 to be detected and all tags lying within the range tr has a detection probability that exponentially decreases as the distance r increases. λ, α and β are parameters that measure detection probability when a target is within a distance of r.

We conduct an experiment to obtain the sensor model in a classroom with enough open space. The RFID device is RRU1861, an UHF RFID integrated with a circularly polarized antenna. The reader is placed at a height of 35cm from the floor while the tags are uniformly distributed onto the floor, which is portioned into 11×11 grids with each tag occupying a grid. The size of grid is 7.5 cm. The detection count of each tag among a total of 2000 scans is recorded. It is found that distance is much more significant in affecting the successful detection than orientation of the tag with respect to the antenna. That is

why only distance is considered in our model. The measurement is repeated by rotating the antenna a bit each time and the results are averaged over different orientations of the antenna with respect to fixed tags. Fig.3.3 shows a visualized probabilistic sensor model. Through a model based fitting using the experimental data, we also obtain the parameters in Equation (3.1), $\alpha = 2.3374$, $\beta = 0.0011$, $\lambda = 0.7921$.

3.4 The Hybrid Method for RFID Reader Tracking

3.4.1 Overview of Our Method

The objective in this chapter is to design an accurate and efficient method for object tracking using the passive RFID. Tracking is continuous localization of the moving object. The frequency of localization is usually determined by the application. Given the localization frequency, the accuracy is measured by the average localization error over the time slots. The efficiency is measured by the average computational time of the localization algorithm in one time slot. The computation time should be less than the required localization period; otherwise, it can not satisfy the real time property of the tracking application. The less the computation time is, the better efficiency the algorithm has. Traditional methods in tracking mainly use two methods, namely Weighted Centroid Localization (WCL) and Particle Filter (PF). WCL is cheap but yields bad accuracy in many application scenario, while PF has good accuracy, but it costs much computational cost. We aims to design a RFID tracking method which has better performance than traditional WCL and PF methods.

Our method includes two simple ideas. In the following, we briefly explain them and illustrate the intuition behind them.

•**Integration of WCL and PF**. We study the accuracy of WCL and PF when they are used in RFID reader tracking. We observe that, one one hand, the accuracy of WCL depends on the velocity of the object. If the velocity is relatively slow, WCL has satisfactory

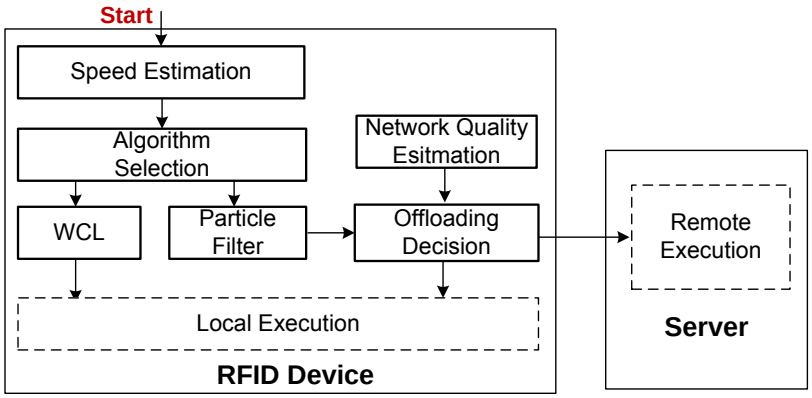


Fig. 3.4: Work flow of the hybrid method

accuracy; otherwise the accuracy becomes very bad. On the other hand, the accuracy of PF is much better, and does not change much with the velocity of moving object. Thus, we ask the question, if we can estimate the velocity of the moving object, can we replace the PF by WCL when the velocity is relatively small in order to save the computation cost, while not sacrificing the accuracy?

•**Computation Offloading**. The mobile RFID device inherently has limited computing capability, and can cause very high computation cost on the device. However, as more and more computation power is integrated into the environmental infrastructures, and there exist abundant wireless techniques connecting the RFID device and infrastructures, can we offload the complex computation involved in PF onto the infrastructures?

With these two questions, we design a hybrid tracking method for better accuracy and efficiency. Fig.3.4 shows the work flow of our method. At each time slot, the RFID devices first estimate the speed of the moving object based on recent RFID readings. After the speed estimation, the component of algorithm selection is performed. This component is responsible for choosing the better algorithm from WCL and PF according to current speed. We have a threshold of speed for the algorithm selection. If the speed is large than the threshold, PF is selected to estimate the location in current time slot; otherwise, WCL

is selected. As WCL is very cheap in computation, if WCL is selected, the algorithm is then executed locally on the RFID device. However, if the PF is selected, the algorithm seeks to offload the computations onto the nearby servers based on the network quality. The component of network quality estimation is performed periodically.

The following subsections explains the details on: 1) how to use WCL and PF into the RFID tracking; 2) how to estimate the velocity of moving object simply using the RFID readings; 3) how to determine the velocity threshold for algorithm selection; 3) and what is principle of offloading computations in PF to the infrastructures.

3.4.2 Weighted Centroid Localization (WCL)

The Centroid Localization (CL) is proposed at first in the outdoor localization for wireless devices [BHE00]. The device to be localized calculates its own position by a centroid determination from all the positions of the beacons in range. Similarly, the CL scheme can be easily used for the indoor RFID positioning system. The target position is estimated by a centroid calculation of all the positions of the tags that have been successfully detected.

However, the CL has two limitations when it is used in RFID positioning system. First, as described in the previous section some tags may be missed even when it is within the reading range. Second, CL method inherently results in a coarse estimation. For limitation one, the failure of identification of tags can be avoided by multiple detection attempts. For limitation two, a weighted CL (WCL) method [BT08][BGGT07][BHE10] is applied to improve the accuracy. The key idea of WCL is to assign greater weight to those tags which are estimated to be closer to the target and less weight to the farther tags. The target position is estimated according to the following equation:

$$l = \frac{\sum_{i=1}^{n}(\beta_i T_i)}{\sum_{i=1}^{n}\beta_i} \tag{3.2}$$

where β_i represents the weight assigned to tag i. The weight should be determined by the

distance of tags from the target according to geometrical rules. In our method, the weight is simply approximated by the count of successful detection during a number of detection attempts. This may be explained by the fact that the detection probability is inversely proportional to the distance. The weight of tag i is given by $\beta_i = \sum_k r_k^i$.

The key idea of WCL is that multiple observations are performed at the target position at the aim of increasing the accuracy. The successful detection count indicates the distance of tag from the target and thus is used to weigh the coordinates of the detected tags. However, the WCL may result in an increasing error when it is applied into tracking mobile object. This is because the target keeps moving when the reader is gathering data. To overcome the above issue, we extend WCL method in time domain for tracking a moving object.

Extend WCL Method in Time Domain

In specific, the current location is estimated by smoothing RFID reading of recent N observations. Historical RFID reading are assigned different weights in estimating current position at time t. Intuitively, the latest RFID reading should be given largest weight in this context. For convenience, these weights are named temporal weights in order to distinguish from weights introduced previously, which are in space domain and called spatial weights. Therefore, the spatial weights of tag i at time t are calculated according to

$$\beta_i(t) = \sum_{k=t-N}^{t} \lambda_t(k) r_k^i, \tag{3.3}$$

where $\lambda_t(k)$ is temporal weight of RFID reading at time k. In our method, temporal weights are determined by intersection of reading range at time k and that of time t.

$$\lambda_t(k) = \frac{\text{Area}(S_t \cap S_k)}{\text{Area}(S_t)}, k = t-N, t-N+1, \cdots, t-1, t \tag{3.4}$$

Through some geometric rules, Equation (3.4) can be simplified by the following equation:

$$\lambda_t(k) = f[\frac{(t-k)v}{2f \cdot tr}], \tag{3.5}$$

where

$$f(\alpha) = \begin{cases} 2[\arccos\alpha - (1-\alpha)\sqrt{2\alpha - \alpha^2}]/\pi, 0 < \alpha < 1; \\ 0, \alpha >= 1 \end{cases} \tag{3.6}$$

v is the current average moving speed. f is the observation frequency of RFID reader. tr presents the reading range of the reader. Equation (3.5)(3.6) shows that, given f and tr, the temporal weight is a decreasing function in terms of the time difference $(t-k)$ and the speed v. The reason by including v into this function is that the faster an object is moving the sooner previous readings will expire. Then the next challenging problem is how to estimate the average moving speed of the object?

How to estimate the object's speed

In this work, we first calculate the time duration $d_i(N)$ that each tag i stays in the reader's reading range in the last N time slots (assume $N > d_i \forall i$). Then we select the maximum one $d_{max}(N) = \max d_i$ to roughly estimate average moving speed as follows:

$$v = \frac{2tr \cdot c}{d_{max}(N)} \tag{3.7}$$

In Equation (3.7), $2tr$ is the diameter of the reading range and $0 < c \leq 1$ is scaling constant depending on the tag density. Above speed calculation is motivated by the following observation: the tag with largest d_i tends to be the one which is closest to the object's actual moving path. Theoretically, if a tag is right on the moving path, it should stay in reader's reading range until the object traveling distance $2tr$. The probability that the closest tag is right on the object's moving path is clearly increased as the density of the tags increases. Therefore c is an increasing function of tag density.

Considerations for Transmission Range tr and Tag Density

Most RFID readers in current market are able to adjust their power level. Assume the reader antenna was circularly polarized, the transmission range tr usually increases as the power level increases. The larger tr is, the more tags will be included into the centroid estimation of the object each time WCL is performed. The question is what is the optimal tr when using WCL method in object tracking. Intuitively, the optimal tr depends on the tag density. Another question is how the tag density affects the accuracy of WCL. Is it true that the more densely the tags are deployed, the more accurate results will be obtained through WCL?

We analyze WCL by numerical simulations featuring a variety of tag densities. The tags are uniformly distributed onto a two-dimension plan according to a regular pattern. Only triangular pattern and square pattern are considered. The tag density is defined as the size of spacing between tags. Spacing in triangular pattern is normalized into that of square pattern, following the rule that the node number per square meter remains the same. The transmission range of the reader tr is preset to a constant value. The speed of the moving object is also preset to a constant, satisfying $v = \frac{2tr \cdot f}{60}$. The tag density increases from $0.2tr$ to a upper bound of $1.4tr$, beyond which there exists blind zone where the object detects no tag. The sensing model of Equation (3.1) is used to generate the simulated RFID readings. The location error is defined as the distance between the exact position and the estimated position of the reader.

Fig.3.5 shows that the location error depends on the tags density. We can have the follow conclusions. (1) The optimal density for WCL algorithm is located in the interval of $(0.8tr, 0.9tr)$. Increasing tag density under the spacing size of $0.8tr$ nearly does not improve the accuracy. However, a sparser density upper the spacing size of $0.9tr$ leads to serious degeneracy of the accuracy. (2) The necessary tr of the RFID reader should be as large

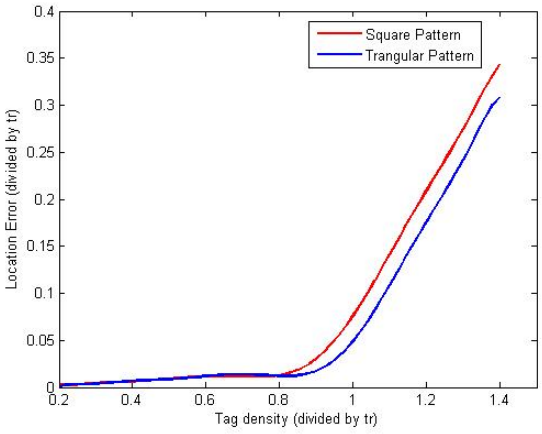


Fig. 3.5: The location error of WCL depending on the tags density

as $1.1a$, where a is the average spacing size between two tags, indicating the deployment density of tags. (3) The deployment pattern of tags, either square pattern or triangular patter, does not affect the accuracy.

3.4.3 Particle Filtering

In this subsection, we solve this object tracking problem through a Bayesian approach [AMGC02]. Using this approach, two models are generally required: first, a model describing the evolution of the object's location with time (the system model); second, a model relating the noisy RFID readings (the measurement model). These models are presented in a probabilistic form. Considering the evolution of the position sequence of a moving object, $l_k = (x_k, y_k, \theta_k)$, $k = 1, 2, 3, \cdots$, we have the following system model

$$x_k = x_{k-1} + \frac{v}{f} \cos\theta_{k-1} \tag{3.8}$$

$$y_k = y_{k-1} + \frac{v}{f} \sin\theta_{k-1} \tag{3.9}$$

$$\theta_k = \theta_{k-1} + U(-\Delta\theta, \Delta\theta) \tag{3.10}$$

where x_k, y_k are two coordinates of the object's location, θ_k is the heading orientation of the object, $U(-\Delta\theta, \Delta\theta)$ is an increase angle of the heading direction which yields a uniform distribution, v is the velocity of the object, f is the observation frequency of the reader. The measurement model is given by

$$p(z_k|l_k) = \prod_{i=1}^{K} p(r_k^i|l_k) \tag{3.11}$$

$$p(r_k^i|l_k) = r_k^i p(d_i) + (1 - r_k^i)(1 - p(d_i)), \tag{3.12}$$

where d_i is the distance of tag i from target and $p(d_i)$ is the RFID sensor model described in Section 3.3. The objective is to recursively estimate l_k from the RFID readings, denoted as $z_k = \{r_k^i\}_{i=1}^K$.

From a Bayesian perspective, the tracking problem is to recursively calculate some degree of belief in the state l_k at time k, given the data $z_{1:k}$ up to time k. It is required to construct the posterior pdf , usually in two stages: prediction and update.

A particle filter approximates the posterior pdf at time k by a set of M particles. Each particle contains a position hypothesis $l_k^i = (x_k^i, y_k^i, \theta_k^i)$ and a weight w_k^i. w_k^i indicates the importance of the i-th particle. A position estimate $\hat{l_k}$ can be obtained by $\hat{l_k} = \sum_{i=1}^{M} w_k^i l_k^i$. The particle filtering algorithm takes the following steps in a recursive fashion:

- **Prediction** The target position at time k is predicated by propagating all particle positions at time $k-1$ according to system model, Equation(3.8-3.10).

  $$l_k^i \sim p(l_k|l_{k-1}^i, v, \Delta\theta)$$

  We assume that v has a Gaussian distribution, $N(v_k, \sigma_k^2)$. The model probabilistically describes the transition of system from old state for a given measurement v_k.

- **Updating** Particles are reweighted according to the latest RFID reading z_k and the measurement model, Equation(3.11- 3.12).

$$w_k^i = w_{k-1}^i p(z_k|l_k^i)$$

The weighted is then normalized by $w_k^i = w_k^i / \sum_{i=1}^{M} w_k^i$.

- **Resampling** A new set of n particles with equal weights $1/n$ is obtained from the old set of particles. The possibility of choosing particle i is proportional to its weight w_k^i. The objective of resampling is to reduce the effects of degeneracy. The basic idea is to eliminate the particles that have small weights and to concentrate on particles with large weights.

The particle filter turns out to be a robust numerical method for the solution of optimal estimation, especially in case of non-Gaussian noisy observations. Before the recursive steps, initial particles are chosen to be uniformly distributed in state space. An increasing particle number usually leads to more accurate estimation. However, particle filter method becomes computationally expensive as particle number increases.

3.4.4 Adaptive Algorithm Selection of WCL and PF

Limitation of WCL

WCL faces two weaknesses in object tracking. First, if the tags are deployed sparsely onto the ground, it is possible that tr can not achieve the necessary value, $1.1a$, even when the Reader's power reaches the maximum level. In this case, the accuracy of WCL will be degenerated seriously, as shown by Fig.3. Second, the observation frequency of RFID f always has an up-bound in order not to affect the detection rate of tags. Given this up-bound of f, if the object moves in a high speed, most of the measurements in the past become invalidate to calculate the current location. Therefore, the accuracy will decrease as the speed of the object increases, which is also demonstrated by Fig.3.7c in Section 3.5.

Limitation of Particle Filter

Although Particle Filter (PF) has been widely used in object tracking for its satisfactory accuracy, the biggest limitation of PF comes from its high computational cost. Table 3.3 demonstrates its costliness in computation. Since mobile objects are usually equipped with resource constrained devices, continuously running PF on these devices may not be possible.

The hybrid method to integrate WCL with PF

Aware of limitations of both WCL and PF, the question here is, can we combine these two methods together in order to achieve both high accuracy and efficiency in object tracking? To realize this objective, our strategy is that in any case WCL do not has worse accuracy than PF, WCL will be adopted to replace PF in object tracking. These cases are formulated by the following conditions: $tr_{max} >= 1.1a$ and $v <= v_{th} = \frac{2tr \cdot f}{N_{th}}$, where v_{th} is the velocity threshold, N_{th} is the required duration (denoted as the number of time slots) by WCL that a tag can stay in the reading range of a moving object.

Usually with the known density of deployed tags, the first condition, $tr_{max} >= 1.1a$, can be easily judged before the runtime. If this condition is satisfied, the power of the reader will be tuned to the level, in which case $tr = 1.1a$. During the runtime, the moving speed of the object is estimated according to Equation (3.7). If the estimated speed is less than v_{th}, WCL will be executed; otherwise, PF will be performed. Instead of deciding which method should be performed before the runtime, we integrate WCL with PF by an adaptive execution of these two methods in the runtime according to the estimated speed of the moving object. This complies with most practical situations that the object moves with a varying speed rather than a constant speed.

Another potential benefit of the hybrid method is that PF is always guaranteed to perform in a relatively high speed. High speed ensures that the system status transition

model is stable. It means, the increase of the heading orientation of the object, presented in Equation (3.10), can be limited into a small interval. It helps increase the particle sampling efficiency by narrowing the area where candidate particles are selected.

Note that N_{th} can affect a lot the performance of the hybrid method. If N_{th} is set too high, that means the threshold of the speed v_{th} is too low, the hybrid method has little improvement on the computational cost compared with particle filter. Oppositely, if N_{th} is set too low, the hybrid method sacrifices the accuracy too much. It is hard to prove theoretically the optimal value of N_{th}. However, we can acquire an optimal value of N_{th} from the numerical simulations. Section V demonstrates an optimal value of N_{th} in a simulated scenario.

3.4.5 Computation Offloading

To further reduce the computation complexity on the RFID device, our method can offload the compute-complex part of the tracking algorithm to nearby server. Although offloading computation onto the service can lower the computational cost on the RFID device, it brings additional communication cost due to the transmission of required data between the RFID device and server. The principle of the offloading decision is that, given the algorithm function, if reduced computation cost by remote execution is large than the incurred communication cost, then it is better to offload this function onto the server; otherwise the function is better to be executed locally.

•**Offloading Principle**. Assume the estimation of location at time t with the RFID readings z_t needs time τ_l if it is performed locally on the RFID device, and needs time τ_r if it is executed remotely on the server. Let d_s denote the size of transmission data in remote execution. Let B denote the bandwidth of the wireless connection between the RFID device and the server. We have the following offloading decision. If $\tau_m - \tau_r > \frac{d_s}{B}$, the algorithm is then offloaded to the server at this time k; otherwise, the algorithm is executed on the

Table 3.2: Local and remote computation cost

Execution time	τ_l	τ_r
WCL	2.7 ms	0.12 ms
Particle filter	215 ms	6.8 ms

RFID device. In the following, we describe how to collect τ_r, τ_l, d_s and B in real systems.

In the hybrid method, the WCL and PF algorithms are adaptively executed according to the object's velocity. We collect both the local and remote computation cost of WCL and PF from our experimental implementation, which is described in details in Section 3.6. The main processing platform of the RFID device in our implementation is the Mica2 sensor mote. The remote server is the Fujitsu LifeBook S Series. Table 3.2 shows the one time computational cost of WCL and PF respectively in location and remote execution. Both the local execution of WCL cause very litter computation cost, i.e., 16.4 ms. For simplicity in our system, in our method we only consider the offloading of PF. Note that the execution time of PF varies depending on the number of particles. In Table 3.2, the execution time of PF is measured when the number of particles is 2000.

To make the optimal offloading decision, we need to estimate the communication cost of PF when it is executed remotely. At each time of location estimation, the PF needs to predict the state of the particles according to the mobility model of the object, and then update the weights of the particles based on the RFID readings in this time slot. Fig.3.6 shows the data transmission between the RFID device and the server when PF is performed remotely. At the beginning of the time slot t, the RFID device uploads the RFID readings z_t to the server. The server does the prediction and resampling, and returns the state of particles as well as the estimated location at time t. The data required to transferred on the network mainly includes the observed RFID readings z_t, the state of particles, i.e., the

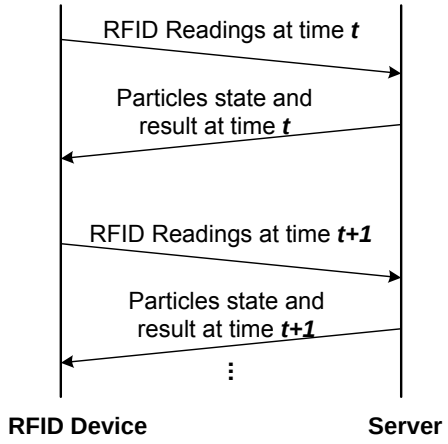


Fig. 3.6: The data transmission between RFID device and server if particle filter is executed remotely

parameters w_k^i and l_k^i. The reason why the state of particles need to be returned to the RFID device is that the wireless network connection may be intermittent. We need to backup the state on the RFID device. In case the network connection is down, the PF can be continued locally with the backup particles state. Note that the size of transmission data increases as the the number of particles in PF increases.

With the size of transmission data, we need to measure the bandwidth of the wireless network, in order to estimate the communication cost. In our method, we periodically estimate the network bandwidth by transferring a 10K file between the device and server. The bandwidth is the testing file size divided by the transmission time. In our experiment, the RFID connects the server through ZigBee motes. We measure the bandwidth fluctuation through carrying with the RFID device and walking around in one classroom. We found that the bandwidth indeed fluctuates. The fluctuation period is on the order of several seconds.

3.5 Performance Evaluation through Simulations

In this section we evaluate the performance of the proposed hybrid method. We consider the following metrics: location error (accuracy) and computational cost (efficiency). First, we evaluate respectively and compare the performance of WCL and Particle Filter (PF). The parameter we consider in the comparison is velocity because it is the main factor that distinguishes the performance of these two methods. Then, we evaluate how much the hybrid method can improve the performance by considering a trajectory along which the object moves with a varying speed.

3.5.1 Performance Comparison between WCL and Particle Filter

The simulated environment is a $4m \times 4m$ plan. The RFID tags are densely deployed onto the plan. The spacing size of tags a is $0.27m$. The reading range of the reader tr is $0.3m$. The observation frequency f is $10Hz$. Assume that the object moves with a constant speed around a rectangle trajectory in the 2-Dimension plan. The RFID model described in Section 3.3 is used to generate the simulated RFID readings. The location is estimated continuously during one trajectory through different tracking methods. The accuracy is presented by an average of the location error at each time slot.

Fig.3.7c compares the location error of WCL and PF under different velocities. We can see that the location error of WCL increases as the velocity increases while the accuracy of particle filter has no association with the velocity. Better accuracy is achieved by PF than WCL when the velocity exceeds $0.2m/s$. Less then the value of $0.2m/s$ both the two methods have almost the same accuracy. Table 3.3 compares the execution time of these two methods. Simulations for both algorithms are run on the same platform (Matlab Environment on a Fujitsu LifeBook S Series). To compare the computational efficiency among the cases of different velocity, the number of trajectories are tuned for each velocity

Table 3.3: Efficiency comparison between WCL and PF

v(m/s)	0.1	0.2	0.3	0.4	0.5	0.6
WCL(ms)	119	86	75	64	65	66
PF(ms)	7593	8484	10568	9792	9260	9276

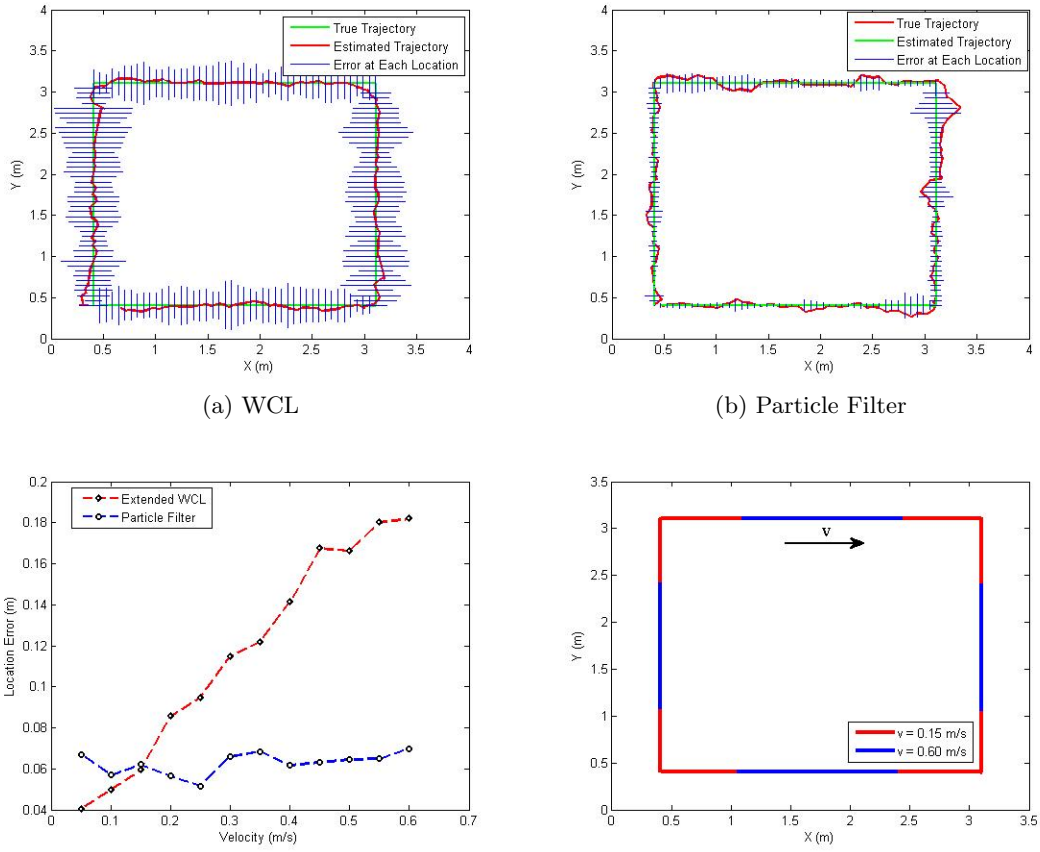
so that the tracking time in all the cases keeps the same. From Table 3.3, we can see that WCL is obviously more efficient than PF whatever the target's velocity is.

Fig.3.7a and Fig.3.7b shows the location error at each time slot as well as the estimated trajectory when the object's velocity is $0.6m/s$. We can see that the location error of WCL is larger than that of PF. It is interesting to note that the location deviation of WCL is along the trajectory while the location deviation of PF is perpendicular to the trajectory. This is because that executing the WCL periodically for object tracking has a critical limitation. If no tag is detected in current time slot, the tags detected in the past time slot will dominate the centroid estimation of the current position. Therefore, the result at current time is very likely to be the real location in the past time slot. In specific, this location error becomes very serious when the object's speed is high.

3.5.2 The Hybrid Method of WCL and PF

We compare the performance of the hybrid method with that of other two methods, WCL and Particle Filter, by considering a more practical scenario where the object's velocity varies during the trajectory. Fig.3.7d shows a simple case that the object circulates around the rectangle with two different velocities alternatively. Around four corners the object moves with a slow velocity, $v = 0.15m/s$. In other intervals the object runs with a fast velocity, $v = 0.6m/s$.

How to select the threshold v_{th} and N_{th} when the hybrid method is performed in the scenario? It has been shown in Fig.3.7c that the optimal v_{th} is $0.2m/s$. Below this threshold,



(a) WCL

(b) Particle Filter

(c) Accuracy depending on the velocity

(d) A Trajectory

Fig. 3.7: Simulation environment and results

WCL has the same accuracy with PF, but above it WCL has much worse accuracy than PF. The N_{th} is computed by $N_{th} = \frac{2tr \cdot f}{v_{th}} = 30$.

Table 3.4 presents the location error and execution time of the three methods. It shows that hybrid method leads to much lower location error than WCL. Moreover, compared with particle filter, the hybrid method saves about 75% computational cost while achieving the same accuracy.

3.5.3 The Offloading Strategy

In this section, we evaluate the influence of offloading strategy onto performance, particularly in term of the efficiency. We compare the performance of the following five tracking

Table 3.4: Performance comparison between the three methods

Schemes	WCL	PF	HybridMethod
LocationError(m)	0.187	0.068	0.062
ExecutionTime(s)	0.08	4.53	1.26

methods.

•**WCL-Local**. In this method, the tracking algorithm is WCL. The algorithm is executed on the RFID device.

•**PF-Local**. In this method, the tracking algorithm is Particle Filter (PF). The algorithm is executed on the RFID device.

•**PF-Offloading**. In this method, the tracking algorithm is Particle Filter (PF). The execution of PF can be offloaded onto the server at each time slot.

•**Hybrid-Local**. In this method, the tracking algorithm is adaptively selected from WCL and PF according to the estimated velocity. The algorithm is executed on the RFID device.

•**Hybrid-Local**. In this method, the tracking algorithm is adaptively selected from WCL and PF. The algorithm can be offloaded onto the server.

In this evaluation, the object's moving trajectory and velocity have the same setting with Fig.3.7d. The time used in this trajectory is $\frac{8}{0.15} + \frac{8}{0.6} = 66.7s$. As the sampling frequency is 10 Hz, we have about 667 time slots in the whole trajectory. The number of particles in all PF executions is set as 2000. Table 3.2 shows that profiling computational cost of WCL and PF in each step (or time slot) respectively on the RFID device and the server. Since the WCL runs very fast on local device, we do not consider the offloading of WCL. When PF is offloaded onto the server, we measure that the size of transmitted data between the RFID device and the service is 5.2 kB.

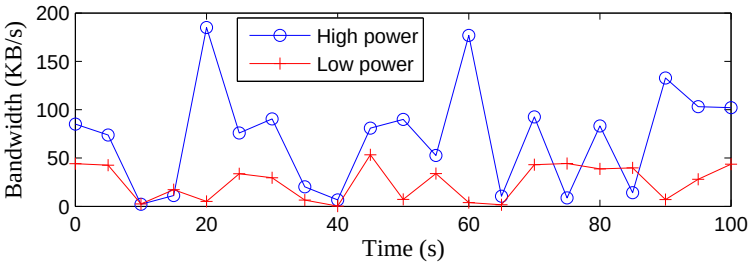
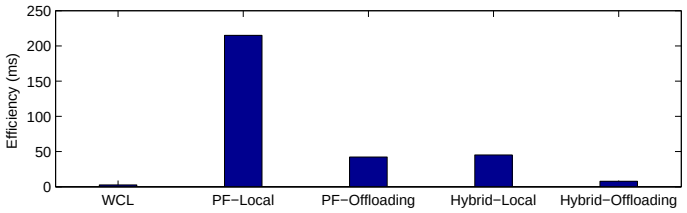


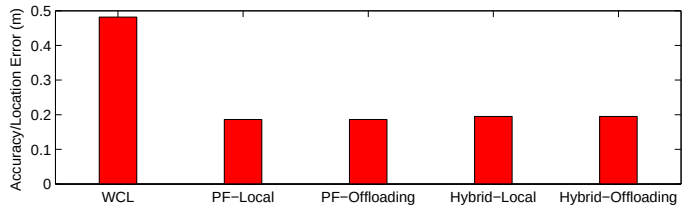
Fig. 3.8: Network bandwidth trace

We collect real network bandwidth traces from one classroom. In the classroom, one notebook is deployed as the server. The notebook is connected with a TelosB mote. The RFID device attached with Mica2 mote communicates with the server through ZigBee. We carry the device and move randomly around the classroom. The network bandwidth is measured by transferring one 10kB testing file. The measurement is done every 5 seconds. Fig.3.8 shows two network traces when the transmission power of the ZigBee motes is set with various levels. In high power setting, both motes have level 256 in transmission power, while in low power setting, both motes have level 128 in transmission power. We use the network trace in high power setting to evaluate the performance with offloading strategy. Note that only the first 66.7s of the whole network trace is used in the evaluation.

Fig.3.9a compares the efficiency of the five tracking methods under this evaluation setting. We can see that both the PF and the hybrid method with the offloading strategy have better efficiency than the corresponding methods without offloading. In particular, the PF-Offloading needs 42.3 ms running time on average at each time slot, while the PF-Local needs 215 ms running time. The offloading strategy brings about 5X efficiency improvement to the PF. The Hybrid-Offloading needs 7.9 ms running time, and is more than 5X better than Hybrid-Local that has 45.2 ms running time per time slot. Among the five tracking methods, WCL and Hybrid-Offloading have much less execution cost than other three methods. Taking the accuracy into account, as shown in Fig.3.9b, the Hybrid-Offloading



(a) Efficiency/Algorithm running time



(b) Accuracy/Location error

Fig. 3.9: Comparison of various methods in terms of accuracy and efficiency

has the best performance among the five methods.

We then evaluate how the computational efficiency of the methods changes depending on the bandwidth of wireless connection between the RFID device and the server. Fig.3.10 plots the efficiency as the bandwidth increases from zero to 300kB. The two methods with offloading strategy, Hybrid-Offloading and PF-Offloading, have decreasing execution cost as the bandwidth increases. This is because under the offloading strategy, large bandwidth leads to short time in the data transmission between the device and server. When the bandwidth is as little as zero, the methods with offloading strategy have the same execution cost with the methods running locally on the device.

3.6 Experiments

We have implemented the RFID based tracking system in two application scenarios, indoor wheelchair tracking, and LRV tracking at one Hong Kong MTR spot. The results are presented as follows.

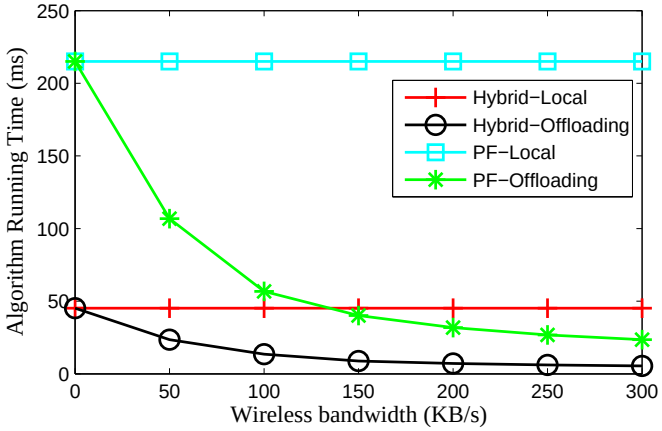


Fig. 3.10: Algorithm running time varies depending on the network bandwidth

(a) (b)

Fig. 3.11: The deployment of RFID system for indoor wheelchair navigation

3.6.1 Indoor Wheelchair Tracking

In the scenario of wheelchair tracking, the RFID tags are densely deployed on the ground in a $4m \times 6m$ classroom, shown in Fig.5. The spacing size of tags (density) is $50cm$. The RFID device we attached on the chassis of the wheelchair is an integrated UHF RFID reader with a circularly polarized antenna. The reader contains 18 discrete power levels. We tune the power on level 12 so that the reading range tr is as far as $60cm$, satisfying the equation $tr >= 1.1a$, where $a = 50cm$.

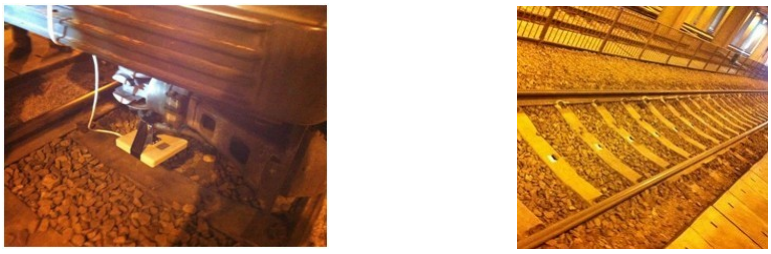
The wheelchair can be controlled with a joystick controller. We attach an extension unit to the joystick for controlling the output signal. This unit is connected to a Mica2 Mote through serial port. The speed of the wheelchair is controlled by a pair of output voltage

Table 3.5: Experiment results for wheelchair tracking

Methods	WCL	PF-O	Hybrid-O	Hybrid-L
Location error (m)	0.482	0.186	0.195	0.193
Time per step (ms)	3.6	89.2	19.4	95.7

values. These voltage values can be set through the Mica2 Mote. In our experiment, the wheelchair moves along a line with a varying speed. During the first half of the entire trajectory, the wheelchair's speed is set to $0.1m/s$. In the second half, the speed is increased to $0.6m/s$. The observation frequency f is $10Hz$. By selecting $N_{th} = 40$, the threshold of the speed for the hybrid method is set to $v_{th} = \frac{2tr \cdot f}{N_{th}} = 0.3m/s$. In the classroom, we deploy a laptop as the server. The laptop is connected with a TelosB mote through the USB, such that the Mica2 mote on the RFID reader can offload computations onto the server through ZigBee protocol. The bandwidth of the ZigBee connection between the RFID reader and server is measured every 5 seconds.

To evaluate the location error, the real location is calculated through the integral of the preset speed. The location is computed periodically using the four tracking methods, WCL-Local (WCL), PL-Offloading (PL-O), Hybrid-Local (Hybrid-L) and Hybrid-Offloading (Hybrid-O). For each method, we repeat the experiment 10 times by choosing different trajectories in the classroom. The result is averaged over the 10 experiments. Table 3.5 presents the average location error and execution time per step of the three methods. It shows that the hybrid method (Hybrid-O) almost achieve the same accuracy as PF (PF-O), but outperforms PF a lot in term of computational cost.



(a) (b)

Fig. 3.12: The deployment of RFID system at one MTR depot

3.6.2 LRV tracking in Hong Kong MTR depot

In the scenario of LRV tracking, we installed an RFID reader below the LRV, shown in Fig.3.12. Note that our RRU1861 URF RFID reader/attenna does not have computation and networking capability. To use it in location estimation, we usually need to attach it with external devices. In the wheelchair tracking experiment, we attach it with the Mica2 mote, while in this experiment, we attach it with one Sharp SH631W smart phone. We deploy our self-made mesh routers along the rail such that all the testing areas are covered by the mesh network. The location can be either calculated locally by the smart phone which is physically connect with the RFID reader, or calculated by the server on the mesh networks.

We deploy a total of 42 RFID tags along the track. When the LRV is moving along the tagged rail, the RFID reader keeps querying for the tags. The observation frequency of the reader f is $20Hz$. We have tested two tracks. On one track, the train is moving at the speed of $15km/h$. On the other track, the train is moving at $35km/h$. We adjust the power level of the reader to the maximum level, level 18, so that the reading range, $tr = 0.8m$, can satisfy the condition required by WCL, which is $tr > 1.1a$, where a is about $0.7m$.

The duration N_d (measured by the number of time slots) that a tag is able to stay in the reading range of the reader is respectively 3 and 7 in case of $v = 35km/h$ and $v = 15km/h$.

For one tag, if all the N detections are failed, this tag will definitely fails to be detected; if any one of the N_d detections is successful, this tag will be detected successfully. We found that the detection rate of tags in the two tests is quite different. The detection rate is about 70% when $v = 15km/h$. When $v = 35km/h$, the detection rate decreases to 45%.

In our experiment, N_{th} is set to an experience value, 30. The LVR's speed in both the two tests are above the threshold. According to the rules of the hybrid method, the location should be estimated by particle filter. In order to evaluate the location error, the real location is also approximated by the integral of the LRV's speed. By performing the PF algorithm based on the data set, we obtains that the location error is $0.45m$ when $v = 15km/h$. However, when v increases to $35km/h$, the location error increases to $1.2m$. We also evaluate the computational cost of the PF algorithm in this LRV tracking application. We compare the computation cost of PF in the case the algorithm is performed locally on the smart phone and the case the algorithm is offloaded onto the server on the mesh network. We find the offloading strategy reduce the computation cost per time slot from 137 ms to 24.8 ms.

This chapter not only provides a hybrid method of object tracking, but also provides a guide for user to deploy the RFID based tracking system under different performance requirements. A tradeoff between the accuracy and efficiency can be achieved by dynamically adjusting v_{th} or N_{th} in the runtime. If the computational cost is critical in the system, a low value of N_{th} is preferred. If high accuracy is desired, a high value of N_{th} is chosen.

In real world, the RFID sensing model will be quite different when the device is performed in different environments. As the sensing model in Section 3.3 is measured in the same configured classroom, it is able to be used into the experiment of wheelchair navigation. However, in the LRV tracking experiment, a more general sensing model, shown in Equation (3.13), is applied into the PF algorithm instead, where p_0 is a constant. Only

when one tag has been detected, the step of filtering out the invalidate particles will be executed according to the sensing model. In other times, the prediction step based on the estimated speed will be executed.

$$p(r) = \begin{cases} p_0, & r < tr; \\ 0, & r > tr \end{cases} \tag{3.13}$$

As shown in the LRVs tracking experiment, neither the PF or the hybrid method is suitable for tracking high speed moving objects cases, the tags detection rate is usually low. The detection rate is as low as 45% when the LVR's speed is $v = 35km/h$. The bad accuracy in this scenario can be explained as follows: 1) when the speed is too high, which also means the value of N_d is too low, our method to estimate the speed will not work. Without information of the speed, the efficiency of particle sampling decreases seriously. 2) The invalidate particles are not filtered out until the next tag is successfully detected. Thus, when the detection rate is low, these unfiltered invalidates particles will leads to large location error.

3.7 Summary

In this chapter we have studied the problem of tracking mobile objects that are able to detect the presence of passive RFID tags at known position of the ground when traveling in their proximity. We proposed a hybrid method for achieving high accuracy and efficiency in object tracking. Through numerical evaluations our method is demonstrated to be more computational efficient than PF while guaranteing the same accuracy with PF. We also implemented two real RFID based tracking systems using our proposed method in two applications, namely, the wheelchair navigation and LRVs tracking at one of the Hong Kong MTR depots. For the former application, our method can achieve better performance than both PF and WCL in either efficiency or accuracy. For the latter application, in most

cases our method can achieve good accuracy, but in some case the LRV moves in a very high speed our method do not perform well due to the low detection rate of the RFID tags.

Chapter 4

Computation Partitioning for Data Stream Applications

In this chapter, we study computation partitioning for data stream application. This chapter is organized as follows. We present an overview of this work in Section 4.1. In Section 4.2, we describe the applications and design requirements. We then present the architectural design in Section 4.3. The application partitioning problem and solutions are presented with details in Section 4.4. Numerical evaluation and real world experiment are presented in Section 4.5 and Section 4.6. Section 4.7 concludes the chapter.

4.1 Overview

By moving the computation to the cloud, many applications which could not be accommodated before due to the lack of significant computation capability and energy power of mobile devices, will be made possible, while leveraging the stable and ample capacity of cloud. We focus on one class of these applications, namely *mobile data stream applications*. These applications usually use camera and/or other high data rate sensors to perform perception related tasks, like face or object recognition, to enable augmented-reality experiences on mobile devices. Specifically, these applications have two characteristics. First, they require continuous processing of high data rate sensors such as camera to maintain

the accuracy. For example, a low frame rate may miss intermediate object poses or human gestures. Second, the computer vision and machine learning algorithms used to process streaming data are often computation-intensive.

However, to make sure that the MCC approach can really bring benefits to both the end users and application providers, we need to address the following two problems.

(1) *Application partitioning problem*: given a mobile application which consists of a set of computational tasks, which tasks should be offloaded to the clouds so that the mobile end users could experience the **maximal performance**? For data stream application, one important performance metric is the speed/throughput that the application processes the streaming data.

(2) *Load scheduling problem*: for the cloud-scale applications, it is possible that a large number of mobile users offload the computational tasks onto cloud at the same time. So how to schedule the offloaded tasks in the cloud so that the **utilization of cloud resources is minimized**? It is critical for the application provider to save their operational cost.

Although the two problems are posed respectively from the requirements of the end users and the application provider, they need to be solved within one system framework. So far, there is no work that treats these two problems jointly. Existing systems [CBC10] [CIM+11] [ZKJG09] [GRJ+09] [RSM+11] that support the application partitioning are only suitable for traditional mobile Internet computing, and do not give any solution on how to use the resources efficiently in clouds to make the applications scalable in cases of serving a large number of mobile users. Other efforts [AAB+05] [CBB+03] [CCD+03] [DG08] [IBY+07] in facilitating large scale cloud applications do not fit well in the MCC applications because they do not support adaptive partitioning of the application between the client and clouds.

In this chapter, we propose a framework for partitioning and execution of the data stream applications under the third MCC approach. The framework contains a novel system

architecture and algorithm which solves the fore-mentioned problem, aiming at achieving maximal performance experienced by the end users and minimal cost in cloud resources favored by the mobile application providers. The main contributions of this chapter are described as follows:

- We design a **system architecture** for the advanced MCC applications. The design is placed on existing mobile platforms and cloud fabrics. The architecture contains two critical mechanisms. First, through online profiling of the characteristics of mobile devices and wireless bandwidth at mobile side, and the back-up of the partitioning results at cloud side, a mechanism is designed to enable fast and adaptive partitioning of the application. Second, a multi-tenancy mechanism is adopted in clouds so that the offloaded computational instances can be shared by multiple mobile users in case they offload the same tasks onto clouds.

- We propose **an optimal partitioning algorithm** for mobile data stream applications. To the best of our knowledge, it is the first work to study the partitioning problem for data stream applications which require parallel execution of different operations onto the streaming data to achieve high processing speed.

- We demonstrate the efficiency of the proposed algorithm through extensive simulations. More importantly, we develop a representative, real world application namely QR-code recognition, and validate the effectiveness of our design through experimental tests on the application.

4.2 Preliminaries

Before describing the design of the MCC system architecture, we first discuss the application that the system can support. The infrastructures that the system is designed based on are then presented. We also point out the design requirements and key methods.

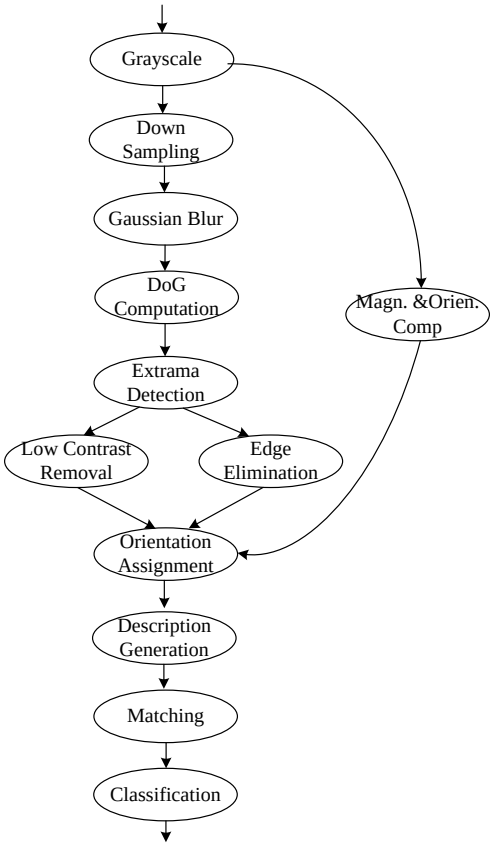


Fig. 4.1: The operations involved in image based object recognition

4.2.1 Mobile Data Stream Applications

Our system targets for the mobile data streaming applications. These applications take the streaming data as input, perform a series of operations onto the data, and then output the results. The input data are sampled periodically from the sensors on the mobile device. Mobile augmented reality is considered as one killer streaming application. The application use the camera and/or other sensors to percept the user's environment/scene, and then augment the original scene with relevant information in the display. The perception is done continuously. The core part of AR applications is the image based object recognition. Fig.4.1 shows the operations involved in the whole process of imaged based object recognition. Note that the SIFT algorithm is used to extract the features [Low04].

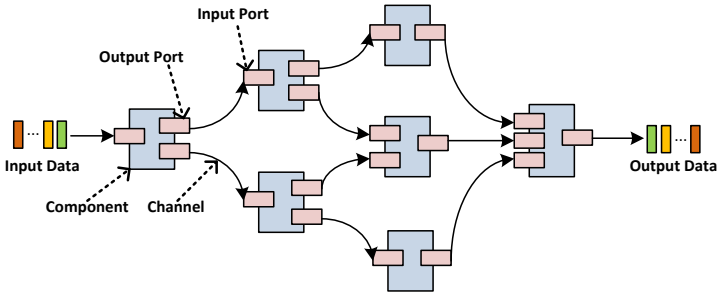


Fig. 4.2: The model for data stream applications

We use a dataflow graph to model the data stream application. The dataflow graph is composed of a set of *components* and a set of *channels* as shown in Fig.4.2. The components run concurrently with each one performing its own functional operations onto the data. The component has *input ports* and *output ports*. Each port is associated with a specific data type. The channel's *capacity* is defined as the maximum number of units of data the channel is able to hold. The channel also indicates the precedence constraint between the operations/components for processing one unit of data, which means the component can not process the data until all of its precedent components complete the operation on that data. The component processing the input data of the application is called the *entry node*. The component generating the output data is called the *exit node*. In real implementations, the components are mapped into threads or processes. The channels are usually implemented by means of TCP sockets, or shared memory or persistent storage. The dataflow model is based on a data centric approach and usually takes advantage of pipeline to accelerate data processing.

4.2.2 System Model

The MCC system model consists of three parts: mobile clients (devices), wireless networks and the cloud (data centers). The mobile client accesses to the Internet cloud services

through wireless networks with limited bandwidth. In the cloud are clusters of commodity servers which are interconnected to each other through high-speed switches. In the following, we describe the terminologies which will be used through this chapter.

End-users and *Application Providers*. End-users refer to the person who consumes the service through their mobile devices that the application provide. Application providers refer to the person/organization who develops, deploys and operates the applications.

Application Instances. Application instance means one execution of the application by one particular end-user. Multiple end-users can run the same application, but each user has its own application instance. Application instances of end-users are different in term of the places hosting the application in the system. For example, assuming an application consisting of three components, c1, c2 and c3, end-user A may run the components c1 and c2 locally, and offloads the component c3 to the cloud, while end-user B may run the component c2 locally and components c1 and c3 remotely. In this case, we say user A and user B have different application instances. Application instance of an end-user may change temporally in the execution place of its components as the mobile device's load and wireless bandwidth varies.

4.2.3 Design Objectives

We have identified the following three requirements for the MCC system design.

High Performance. Two measures are commonly used to evaluate the performance of data stream applications, *makespan* and *throughput*. *Makespan* is the time used to process a single unit of data. It represents the responsiveness of the application. *Throughput* is the rate at which the input data is processed, and it determines the quality of result of the application. Taking an example of the gesture recognition application, the recognition accuracy will be better if the application can process more frames in one second. The MCC system is expected to provide maximal throughput for each end-user while satisfying the

constraint of make-span.

Low Operational Cost. In a MCC system, the resources are provisioned at clouds to accommodate the computation offloaded from the end-users. The resource cost for operations is another critical factor we need to consider in the system design. Assuming that the application provider leases the cloud resources to host the application, the operational cost is measured by the amount of the leased cloud resources. It is required that the MCC system guarantees the minimal cost under given loads from end-users.

Adaptivity and Elasticity. The key benefit of the mobile cloud system is its combinational property of adaptivity and elasticity. Adaptivity means computations (or loads) that are offloaded from the end-user's mobile device to the cloud are adaptive to the end-user's changing mobile environment. For example, when the end-user's device has a high CPU load and good networking bandwidth, most computations may be offloaded onto clouds; conversely, when the end-user's device has an idle CPU and its networking bandwidth is low, most computation may be executed locally. Elasticity means the cloud resources can be provisioned cost-efficiently to meet the end-users' offloading loads. The two properties make our proposed MCC system unique compared with existing system providing mobile services or cloud services.

To achieve the above goals we develop two key techniques, namely Adaptive Computation Partitioning and Multi-tenancy Component as a Service (CaaS). In the first technique, the device characteristics and networking bandwidth are profiled online on the user's device. The partitioning algorithm is triggered on cloud as long as the variance of these profiling parameters exceed the threshold. The technique enables the mobile user to achieve an optimal partitioning whenever the user's environment changes. The second techniques allow the application instances of different users to share the same component on cloud. Because of the component sharing, we do not need to separately allocate resources for each user

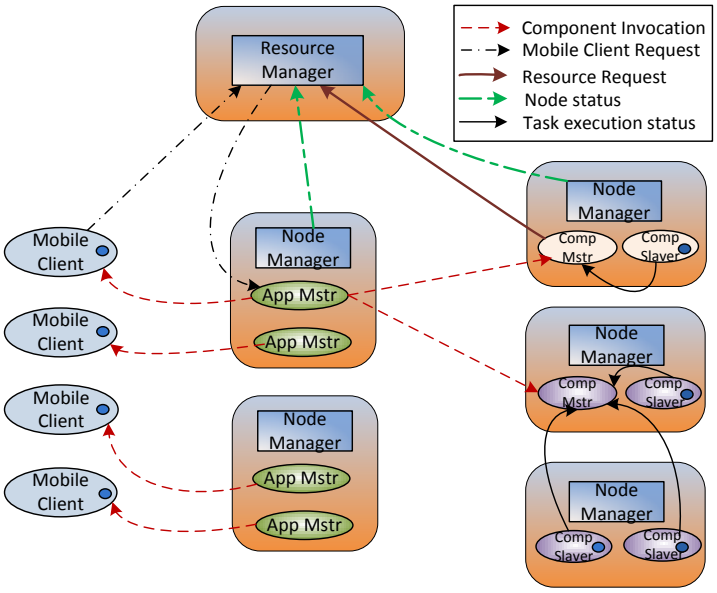


Fig. 4.3: Overview of the application framework

to accommodate everyone's peak data rate, but only need to allocate the resources such that it can serve the peak of the total data rate. Besides, this technique can avoid the frequent loading and unloading of some 'hot' components. Hot means those components are offloaded onto cloud by a majority of the mobile users.

4.3 Architectural Design

Fig.4.3 shows the overview of a dataflow execution framework in mobile cloud computing. The runtime framework consists of software modules on both the mobile side and the cloud side. The client side monitors the CPU workload and networking bandwidth. When the application is launched on the mobile client, an request is sent to the *Resource Manager* in cloud for augmented execution. The resource manager then assigns an *Application Master* to handle the request. The application master first asks the mobile client for its device characteristics such as CPU capability p, its workload η, and the current network bandwidth B. Using these dynamic information from mobile device as well as the static application properties stored in cloud, the application master then generates an optimal partitioning

result, which is presented in Section 4.4. The components assigned to the client are initiated as threads on the mobile device. Other components assigned to the cloud are invoked as services, namely *Component-as-a-Service (CaaS)*. The application master is also in charge of the data transmission between the mobile client and cloud.

In the framework, every mobile application has an Application Master in cloud to augment its execution. The components are shared and invoked by applications as a service in cloud. *Resource Manager* and the per-machine *Node Manager*, which monitors the processes on that machine, constitute the computation fabric in the cloud. *Resource Manager* manages the global assignment of computing resources to Application Masters and CaaSs through cooperation with *Node Managers*.

4.3.1 Adaptivity of Partitioning

The application master, in the middle of the mobile clients and the cloud CaaSs, has two distinct functionalities: (a) to determine an optimal partition results and make the partitioning adaptive to the mobile client's varying environment (local CPU load and wireless networking bandwidth); (b) to coordinate the distributed execution of the dataflow application.

Fig.4.4 shows the software modules on both the mobile client and application master, which provides support for the adaptive partitioning. It is assumed that two logical communication connections exist between both sides: an "always-on" connection but low data rate wireless connection which is for transmitting the control message; another wireless connection with bandwidth B, which is to pipeline the data streams between the mobile client and cloud. The *profiler* on the mobile client measures the device's characteristics at startup and continuously monitors its CPU workload and wireless network bandwidth. The controller on mobile client side maintains some thresholds on the variance of the profiling parameters. If any of the parameters increases/decreases by a value exceeding the threshold, a request

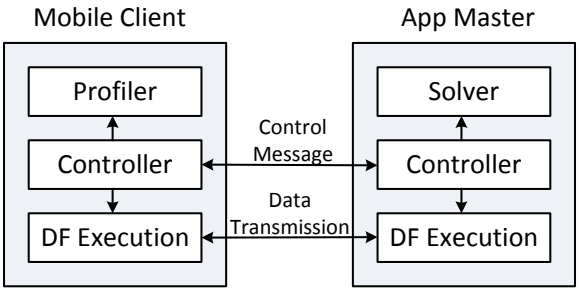


Fig. 4.4: Cooperation between the mobile client and the application master

for updating the partitioning result will be sent to the controller on the application master. The controller of application master calls *optimization solver* to generate a new partitioning result. The optimal partitioning algorithm will be described in Section 4.4. Taking the result as the input, the underlying module *DF Execution* provide runtime support for the distributed execution of the dataflow application.

In the design of our framework, we make sure that the runtime software will not bring much burden onto the mobile device and should be as lightweight as possible. So we put the optimization solver on the cloud rather than the mobile device to reduce the local resource utilization. Although the design feature requires an always-on connectivity, it is reasonable because unless there is wireless connectivity, all the components of the dataflow application is executed locally by default without the need to call the optimization solver.

Besides, the partitioning results for different mobile environments are able to be backed up in the cloud storage. If the request for updating the partitioning has similar input parameters as previous ones, the partitioning result will be directly queried from the back up storage instead of being computed by the optimization solver. The back up mechanism reduces the latency of the partitioning.

4.3.2 Distributed Execution

Fig.4.5 shows the distributed execution of dataflow example withtwo partitioning cases. In the framework, the local components run as threads on mobile device while the remote

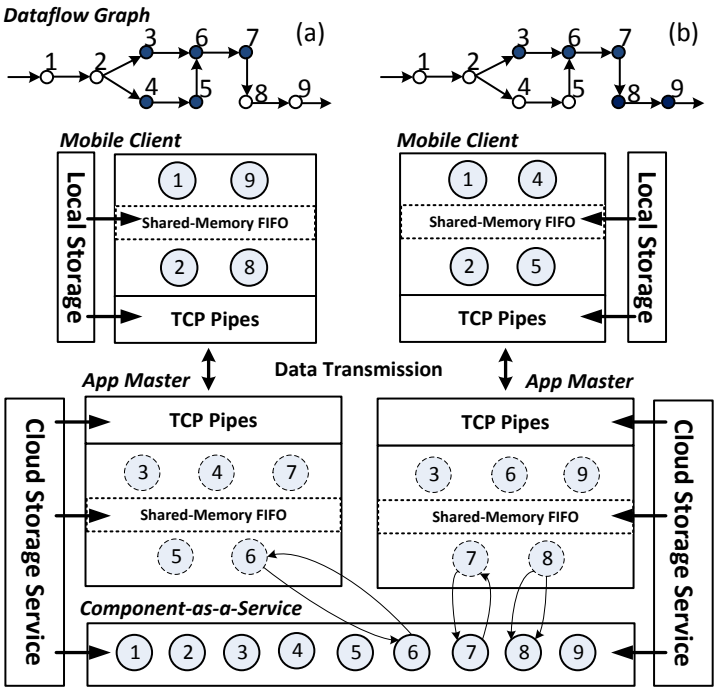


Fig. 4.5: Distributed dataflow execution

components are executed through the invocation of CaaSs. In a partitioned dataflow application, we name the component allocated onto mobile device as *local component*, and the one offloaded onto cloud as *remote component*. The application master has one thread for every remote component. These threads are responsible for data transmission as well as CaaS invocation. Since the threads serve as the images of the remote components, we call them as *image components*.

In a partitioned dataflow graph, the shaded node represents the remote component; the blank one is the local component. The channels are classified into two categories, *crossing channel* and *internal channel*. The crossing channel, e.g. $(2, 3)$, $(2, 4)$, $(7, 8)$ in graph (a), refers to the edge in the graph which connects a local component and remote component while the internal channel connects two local components, e.g., $(1, 2)$ in graph (a), or two remote components, e.g. , $(4, 5)$ in graph (a). The crossing channels are implemented by TCP pipes. Through the TCP pipe, the data is pushed from one component to its successor.

Each TCP pipe has one in-memory FIFO at the receiver side to buffer the data that may not be processed. The internal channels are implemented by shared memory FIFOs. As a result of the FIFOs on all the channels, our framework enables an asynchronous and loosely decoupled way to execute the concurrent components.

4.3.3 Multi-tenancy CaaS

We realize multi-tenancy feature for the CaaSs to allow multiple tenants or application instances to share the component. As shown in Fig.4.3, the multi-tenancy CaaSs implementation adopt a master and slave architecture, in which *Component Slaves* are real entities to do the computation and *Component Master* takes charge of scheduling tenants' loads onto the component slaves. Specifically, *Component Master* negotiates resources from Resource Manager and work with Node Managers to launch/terminate component slaves according to the current request load. The purpose of the multi-tenancy CaaS is to guarantee an elastic utilization of underlying resources to accommodate the scalable CaaS requests.

In our framework, the end users have different application instances even when they run the same application. Each application instance consists of a set of components. The CaaS at the cloud side is usually shared by multiple application instances. According to the partitioning mechanism, application instances have various load requirements on one specific CaaS. The load requirements mean how fast the CaaS is required to process the input data stream.

In order to save the resources, we need to solve the load scheduling problem. The problem is to schedule various loads from the application instances onto the component slaves, such that the number of utilized component slaves are minimized. We assume that the component slaves have the same capacity. The load scheduling problem can be modeled as a Online Bin Packing Problem [SR02].

4.4 Optimal Partitioning Algorithm

In this section, we describe the models, formulation and algorithm for solving the computation partitioning problem. The problem is formulated as an optimization problem, and the proposed algorithm is executed online by the *Application Master* shown in Fig.4.3.

Application Model. The application is modeled as a specific dataflow graph $G = (V, E)$, where $V = \{i | i = 1, 2, ...v\}$ represents its components and $E = \{(i, j) | i, j \in V\}$ represents the dependency between the components. s_i is the average number of CPU instructions required by component i to process one unit of data. $d_{i,j}$ presents the amount of data required to be transmitted on the channel (i, j) for one unit of data. The weight on a node i, denoted as w_i, represents the computational cost (time). The weight on an edge denoted as $c_{i,j}$ is the communication cost (time). Both w_i and $c_{i,j}$ are measured by one unit of data.

Throughput Model. Here, the throughput of the application is the objective for optimization. We define *critical component/channel*, which represents the component/channel that has the greatest weight among all the components/channels. Assuming that all the channels' capacity is unlimited and whatever level of pipeline parallelism is allowed, the throughput of the dataflow application is determined by the critical component/channel, which have the slowest speed to compute/transfer the data. So we have the formula for throughput $TP = \frac{1}{t_p}$, where

$$t_p = max\{\max_{i \in V}(w_i), \max_{(i,j) \in E}(c_{i,j})\}. \tag{4.1}$$

Offloading Model. The offloading decision is made mainly depending on the local computing resources and the wireless networking quality. A few parameters are introduced to model these properties. p is the CPU's capability of the mobile device, measured by the number of performed instructions per second. η is the percentage of the ideal CPU

resource. It also indicates the current working load on the mobile device. So the available CPU resource on the mobile device is ηp. B is the bandwidth of the wireless network for the mobile device to access the Internet cloud. We have the following assumptions in our system model. i) The components running concurrently on the mobile devices are allocated equal CPU resources. ii) If a component is offloaded onto cloud, other components running on the mobile client will speed up because of the acquisition of the released CPU resources. The *speedup factor* is $\frac{N}{N-1}$, where N is the number of components on the mobile device before the offloading event. iii) The cloud always has abundant resources to accommodate the offloaded components such that they will not become the critical component in the dataflow graph. iv) The total wireless bandwidth B are shared by all the *crossing channels*, where crossing channel in the dataflow graph is defined as the one which connects two components residing two sides of different resources. It is possible allowed for the mobile device to allocate disparate bandwidth to different crossing channels. We do not distinguish between the uplink and downlink bandwidth in our model. v) If interdependent components are offloaded onto cloud, the channels connecting between them in the cloud will not become the critical channel. vi) The input data of the application is acquired from the sensors on the mobile device, and output data should also be delivered to the mobile device.

Problem Formulation. Given the dataflow application $\{G(V, E), s_i, d_{i,j}\}$, the mobile device properties $\{p, \eta\}$, and the wireless network bandwidth B, the partitioning problem in this study is the problem of allocating a set of v components of the dataflow graph to the resources (the mobile client and the cloud) and allocating the limited wireless bandwidth B to the potential crossing channels such that the throughput of the data stream application

is maximized. The optimization problem is formulated in Equation 4.2.

$$\begin{aligned}
&\max_{x_i, y_{i,j}} TP = \frac{1}{t_p}, i, j \in \{0, 1, \cdots, v+1\}, \text{ where} \\
&t_p = \max\{\max_{i \in V}(x_i \cdot \frac{s_i}{\eta p} \sum_{i \in V} x_i), \max_{(i,j) \in E}(\frac{d_{i,j}(x_i - x_j)^2}{y_{i,j}})\}, \\
&s.t. \begin{cases} \sum_{(i,j) \in E} y_{i,j}(x_i - x_j)^2 = B, \\ y_{i,j} > 0, \\ x_0 = 1, \\ x_{v+1} = 1, \\ x_i = 0 \text{ or } 1, i \in \{1, 2, \cdots, v\} \end{cases}
\end{aligned} \tag{4.2}$$

The core variables are x_i and $y_{i,j}$. x_i is either 0 or 1 integer, indicating the offloading decision for component i. If x_i equals to 1, component i is executed on the mobile device; otherwise $x_i = 0$ means running on the cloud. $y_{i,j}$ is the wireless bandwidth allocated to the channel (i, j). Note that two virtual nodes, 0 and $v + 1$, are created to satisfy the constraint that the input/output data of the application should be from/delivered to the mobile device. Two edges $(0, 1)$ and $(v, v + 1)$ are added into the set of edges E of the dataflow graph, where node 1 is the entry node and node v is the exit node. Accordingly, $d_{0,1}$ is the size of an unit of input data. $d_{v,v+1}$ is the size of an unit of output data.

Algorithm Design. The objective function shown in Equation 4.2 depends on two variables, x_i and $y_{i,j}$. We first study the problem of allocating the wireless bandwidth B to the crossing edges given a specific partition. It is not difficult to prove theorem 1.

Theorem 1: Given a partition $X = \{x_i | i = 1, 2, \cdots, v\}$, the throughput is maximized

when $y_{i,j}$ satisfies the condition that

$$\begin{cases} y_{i,j} = \frac{d_{i,j}}{t_{comm}(X)}, \forall (i,j) \in E \text{ and } x_i \neq x_j, \\ y_{i,j} = 0, \forall (i,j) \in E \text{ and } x_i = x_j, \\ y_{0,1} = (1 - x_1)\frac{d_{0,1}}{t_{comm}(X)}, \\ y_{v,v+1} = (1 - x_v)\frac{d_{v,v+1}}{t_{comm}(X)}, \end{cases} \tag{4.3}$$

where $t_{comm}(X)$ is the communication cost/time that each crossing channel needs to transfer a unit of data,

$$\begin{aligned} t_{comm}(X) = \frac{1}{B}[(1 - x_1)d_{0,1} + (1 - x_v)d_{v,v+1} \\ + \sum_{(i,j) \in E} (x_i - x_j)^2 d_{i,j}]. \end{aligned} \tag{4.4}$$

So, the original problem can be reduced into

$$\max_X TP = \frac{1}{\max\{t_{comp}(X), t_{comm}(X)\}}, \tag{4.5}$$

where $t_{comp}(X) = \max_{x_i \in X}\{x_i \frac{s_i}{\eta p} \sum_{i=1}^{v} x_i\}$ and X is an v-dimension vector of 0 and 1. The application throughput is constrained either by the speed that the local components process the data or by the speed the crossing channels transfer data.

We then propose a genetic algorithm to solve the reduced partition problem as shown in Algorithm 1. Essentially, we treat different partitions as a population of individuals with different chromosomes. Individuals with higher fitness as defined by Equation 4.2 in this work are more likely to survive and multiply. A partition X is represented by binary string $X = \{x_1, x_2, \cdots, x_v\}$. The encoding method successfully identifies the allocation of either on the mobile device side or otherwise. For example, if x_i equals to 1, component i is executed on the mobile device; otherwise it should be put in the cloud.

The evolution starts from a randomly generated population (line 1). $NIND$ represents the number of the individuals of the population. In each generation, the fitness of every individual in the population is evaluated (line 4-5). Individuals are probabilistically selected from the current population for breeding according to their fitness (line 6). We use roulette

Algorithm 1: The genetic algorithm for partitioning

```
   Input  : NIND, GGAP, MUTR, MAXGEN
   Output: The optimal partition
 1 Chrom ← RandomlyCreatePopulation(NIND);
 2 Gen ← 0;
 3 while Gen < MAXGEN do
 4 |   ObjV ← GetThroughputOf(Chrom);
 5 |   FitnV ← Normalize(ObjV);
 6 |   SelCh ← RouletteWheelSelect(Chrom, FitnV, GGAP);
 7 |   i ← 1;
 8 |   while i < GGAP * NIND do
 9 |   |   CrossingOver(SelCh[i], SelCh[i + 1]);
10 |   |   i ← i + 2;
11 |   end
12 |   for i ← 1 to GGAP * NIND do
13 |   |   SelCh[i] ← Mutation(SelCh[i], MUTR);
14 |   end
15 |   ObjVSel ← GetThroughputOf(SelCh);
16 |   Chrom ← Reinsert(Chrom, SelCh, ObjV, ObjVSel);
17 |   Gen ← Gen + 1;
18 end
19 ObjV ← GetThroughputOf(Chrom);
20 i ← GetIndexOfMaximum(ObjV);
21 return Chrom[i];
```

wheel selection in our algorithm. The probability that each individual is selected is proportional to its fitness. Generation gap $GGAP$ is a control parameter of our algorithm, which represents the number of selected individual divided by the current population size. The selected individuals are modified through crossover (line 7-11) and mutation (line 12-14), and added into the current population. Our generic algorithm evaluates the fitness of all individuals and selects the best ones with constant size of population (line 16). The best individuals serve as the new generation of partitions, which is then used in the next iteration of the algorithm. The algorithm terminates when the number of generations has reach certain upper bound $MAXGEN$. In the last generation, the partition with the highest throughput is chosen as the final partition (line 19-21).

Recall that in each round, our algorithm modifies current individuals using crossover and mutation. Crossover generates new individuals by combining two randomly selected individuals (partitions), say A and B. During crossover, a randomly chosen gene position

divides the binary string of A and B in two parts. One new individual obtains the first section of string from A and the second section of string from B. The second new individual obtains the inverse genes. Mutation takes the string of an individual and randomly changes one or multiple values. The mutation rate $MUTR$ is defined by the ratio of the amount of the changed bits to the total amount of bits in one chromosome.

4.5 Numerical Evaluation

We evaluate the proposed partition algorithm in this section. The performance metric we consider in the evaluation is the throughput of the data stream application.

4.5.1 Methodology

First, we evaluate how the controlling parameters of the algorithms, $NIND$ and $GGAP$, affect the performance. Second, we study the effect of the input parameters of our algorithms including application graphs, the wireless networking bandwidth B and the available computing resource at mobile device ηp. At last, we demonstrate the factor that can affect the computational cost of our algorithm.

The input application graph we consider is the randomly generated application graphs. We have implemented a graph generator to generate the weighted streaming application graphs. We use the level-by-level method to create the graph which was proposed by Tobita and Kasahara[BHE02]. We could control the graph that we want to generate through the following parameters: 1) number of nodes; 2) average out-degree; and 3) communication-to-computation ratio (CCR), where CCR is defined as the ratio of the average communication time to the average computation time as shown in equation (6). If an application graph's CCR is high, it can be considered as a communication-intensive application; otherwise, it

Table 4.1: Configuration in each simulation

No.	NIND	GGAP	MUTR	G(v, d_{out}, CCR)	B	ηp
1	30	*	0.02	G(30, 3, 1)	1	1
2	*	0.8	0.02	G(30, 3, 1)	1	1
3	80	0.9	0.02	G(*, 3, 1)	1	1
4	80	0.9	0.02	G(50, 3, *)	1	1
5	80	0.9	0.02	G(50, 3, 1)	*	1
6	80	0.9	0.02	G(50, 3, 1)	1	*

is an computation-intensive application.

$$CCR = \frac{[d_{0,1} + d_{v,v+1} + \sum_{(i,j)\in E} d_{i,j}]/[(e+2) \times B]}{(\sum_{i\in V} s_i)/(v \times \eta p)} \tag{4.6}$$

We have done a group of simulation to evaluate the effect of both controlling parameters and input parameters to the performance. Table 4.1 shows the configuration of our simulations. In each simulation, we choose one parameter as the variable, which is indicated by '*', while assigning other parameters as constant values. For example, in No. 5 experiment, we study the effect of wireless bandwidth B onto the performance. In our configuration, we treat ηp as one single parameter which indicates the available CPU resources on mobile device. Note that B and ηp shown in the table is a normalized value.

4.5.2 Results

We first presents the result of how the controlling parameters, $NIND$ and $GGAP$, affect both the throughput and the number of iterations that the genetic algorithm needs to converge. Fig.4.6a shows the throughput value in each iteration of our algorithm for different $NIND$s. The configuration of other parameters is shown in Table I under row No.1. We can see that larger $NIND$ value leads to better result. The algorithm takes fewer iterations to converge to the final throughput in case of higher $NIND$ value. Fig.4.6b shows the effect of $GGAP$ on the performance. Larger GGAP value has both better convergence

speed and the final throughput. It indicates that if more individuals are selected from the population for breeding in each iteration, the genetic algorithm will take fewer iterations to find the optimal individual.

We then present the effect of the input parameters on the performance. It contains the application graph properties (graph size v and CCR), ηp and B. The results are compared with other two intuitive strategies with no partitioning: 1) running all the nodes of the application graph on the cloud; 2) running all locally on the mobile device. At first, we study the relationship between the throughput and application graph size v. The configuration parameters are shown in Table 4.1 under No.3 Row. We increase v while keeping the average node's computational cost and communication cost of the edges not variable. So v actually indicates the overall computational complexity of the application. Fig.4.6c shows that our algorithms can typically achieve more than 2X better throughput than other two strategies. It also shows that given the resources B and ηp, the application performance of the partitioning scheme goes down as the application size rises up. When the application size becomes very large, our method tends to have the same performance with the all-cloud method. It is because when the application becomes extremely computational complex, offloading all the nodes of the application graph onto cloud can save much more computational cost than the overhead of communication, while partitioning method has little improvement on the performance in this case.

Fig.4.6d and Fig.4.6e respectively shows the influence of networking resource B and computing resources of the mobile device ηp. It is interesting to find that our method always achieve better performance as B increases while on the other hand, increasing ηp does not necessarily lead to better performance. The different results can be explained as follows. For an optimal partition, the total throughput of the application is either limited by the computation or the communication as explained in Equation 4.5. In the former case

we say the bottleneck is at computation while in the latter case we say the bottleneck is at communication. If the bottleneck is at communication, increasing bandwidth B is definitely able to improve the performance; otherwise, we could always reduce the computation overhead t_{comp} by moving one node from mobile to cloud. Normally this moving operation is likely to increase the communication overhead t_{comm}, but we have the increasing B to accommodate the extra communication overhead. That's why the increase of B usually leads to the raising of the overall throughput. However, it is not guaranteed to reduce the communication overhead t_{comm} by moving one node from cloud to mobile when the bottleneck is at communication. So in this case the increase of ηp can not improve the overall throughput. Fig.4.6e indicates only in the case where ηp is large enough, the throughput of our method sensitively increases as the ηp increases, because in this case the result of our method approximates the 'all-mobile' method.

Fig.4.6f shows how CCR affects the performance. We obtain different CCRs in our simulation by changing s_i/d_{ij} while keeping B and ηp as constant. It shows as the CCR rises the performance of our method first goes down and then rebounds. When CCR is large, running all nodes on the mobile side approaches the optimal performance. Fig.4.6g presents the number of nodes allocated onto mobiles device by our method in case of different CCRs. Obviously more nodes are executed on the mobile device when the CCR increases.

At last, we study the computational cost of our algorithms. Given the internal parameters such as $NIND$ and $GGAP$, the computation cost is measured by the number of generations that our algorithm demands to converge. In practice it is not necessary, if not impossible, to cost a lot of computation to achieve the theoretical optimal throughput. We usually take a critical point, for example 90 percentage of the optimal value, as the actual convergence point. Fig.4.6h shows the generations required to achieve the convergence point for different application graph size v. Our algorithms have larger computational cost as the

application graph size increases.

4.6 Experimental Evaluation

We conduct a series of experimental tests on real-world applications to validate the results. We take one simple application as our test examples: QR code recognition. The application is re-written using the Flow-based Programming (FBP) model [PM10], in which application is modeled as a set of functional components running in parallel and a set of channels streaming data from one component to another.

4.6.1 QR-code Recognition

We spend more than one month to program the QR-code recognition application with the FBP model. The application consists of three phases: image capturing, image pre-processing, and QR-code decoding. In the program, we decompose the three phases into 9 functional components, which are shown in Fig.4.7. For convenience of description, we label each component with a circled number. ⓪ and ⑧ are respectively the entry and exit component of the program. We measure the size of data that are transferred between the components with a 640×480 (300K Bytes) input image. The results are shown by the labels on the edges in Fig.4.7. Note that the size of data transferred between components depends on the size of input image. In our experiment, all the tests use the 300K Bytes input image.

4.6.2 Experiment setup and Results

In the experiment environment, we use the Motorola MB510 Android phone as the mobile device. The cloud resources contain a cluster of PCs in our lab. The wireless connections between mobile and cloud are through WLAN or 3G. The open source runtimes JavaFBP [PM10] are deployed on both the Android phone and cloud nodes to support

Table 4.2: Local computational time of components

Component No.	①	②	③	④	⑤	⑥	⑦
Time (ms)	80	130	110	50	40	30	280

Table 4.3: Transmission time between the components

Edges	Trans. Time	Edges	Trans. Time
⓪ → ①	10240 ms	③ → ⑥	427 ms
① → ②	1280 ms	④ → ⑦	10 ms
② → ③	427 ms	⑤ → ⑦	2 ms
③ → ④	427 ms	⑥ → ⑦	2 ms
③ → ⑤	427 ms	⑦ → ⑧	1 ms

the execution of the FBP programs. Java Message Service (JMS), a Message-oriented Middleware, is also installed together with JavaFBP on both the mobile and cloud nodes to take charge of the data transmission between the distributed components.

First, we profile the computational cost of each component on the mobile device. We run the QR-code recognition program on the Android phone for 30 times. Table 4.2 shows the average time that each component needs to process one 300K Bytes image. Then, we measure the communication cost between the components. It is equal to the data size between two components divided by the wireless bandwidths. Table 4.3 shows the data transmission time between two dependent components under the bandwidth 240 Kbps.

Now we start the system, and demonstrate how the partition changes as the wireless bandwidth varies. Table 4.4 records the partitions under various wireless bandwidths. We can see that when the network bandwidth is as low as 40Kpbs, all the components except ⑦ are executed on the Android phone. As the bandwidth increases, the optimal partition includes more components running on the cloud side.

We also compare the performance of the partitioned application with other two strategies without partitioning. From the results shown in Fig.4.8, we conclude that for the QR-code

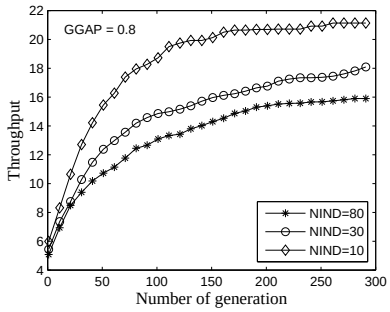
Table 4.4: Partitions under different bandwidths

Bandwidth	Partitions
40 Kbps	Mobile:{①, ②, ③, ④, ⑤, ⑥}, Cloud:{⑦}
240 Kbps	Mobile:{①, ②}, Cloud:{③, ④, ⑤, ⑥, ⑦}
1.2 Mbps	Mobile:{①}, Cloud:{②, ③, ④, ⑤, ⑥, ⑦}

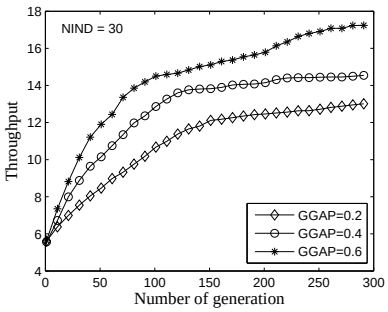
recognition application, the partitioned execution can achieve at least 2X better throughput in reality than the executions without partitioning. For example, when the bandwidth is 240Kbps, the application with optimal partition can process 2.4 images per second, while the application throughput is 0.5 images per second if all the components are executed on the mobile device, and approximates to 0.1 if all the components are executed on the cloud.

4.7 Summary

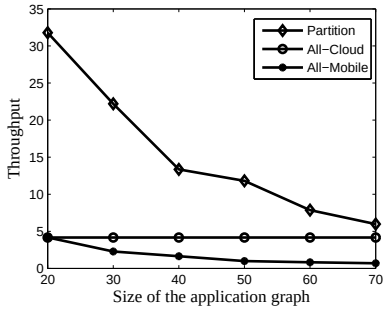
In this chapter we study the computation partitioning problem for mobile data stream applications. We have designed a cloud-based framework to provide runtime support for the adaptive partitioning and distributed execution of such advanced mobile cloud applications. The framework is able to serve large number of mobile users by leveraging the elastic resources in existing cloud infrastructures. Under this framework, we also have designed a genetic algorithm to solve the partition problem. Both numerical evaluation and real world experiment results show that our method can provide more than 2X improvement in the application performance over the methods without partitioning.



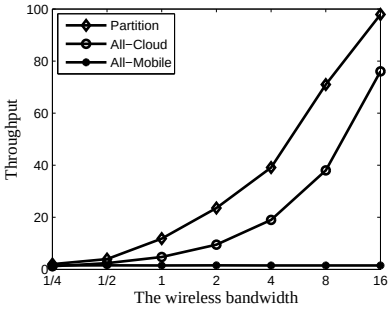
(a) Throughput - NIND



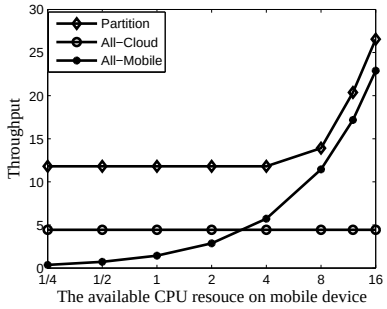
(b) Throughput - GGAP



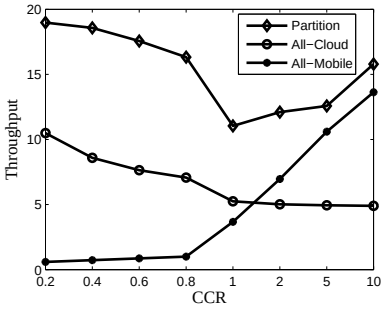
(c) Throughput - v



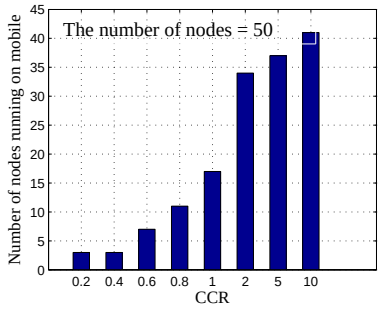
(d) Throughput - B



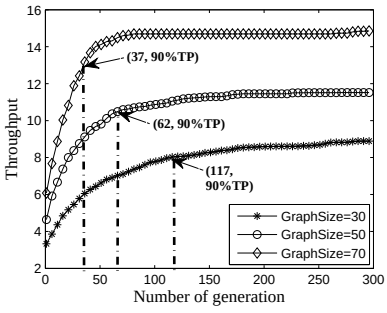
(e) Throughput - ηp



(f) Throughput - CCR



(g) The number of nodes on mobile



(h) Computational cost

Fig. 4.6: Numerical evaluation results

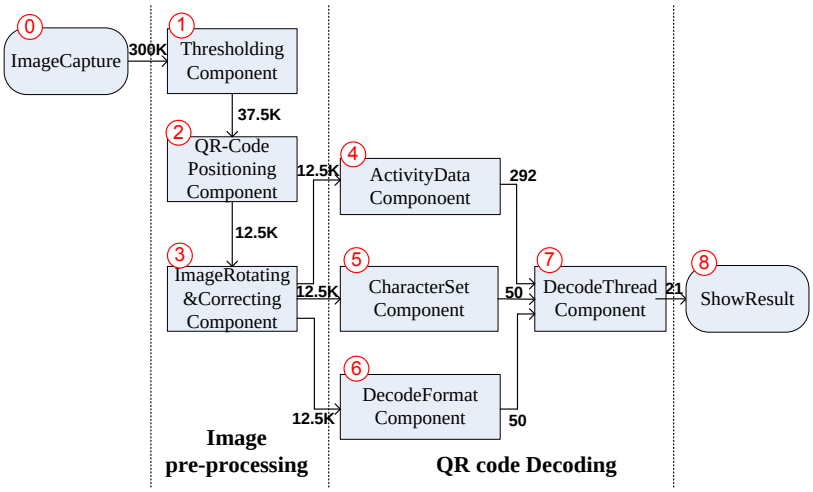


Fig. 4.7: FBP implementation of QR-code recognition

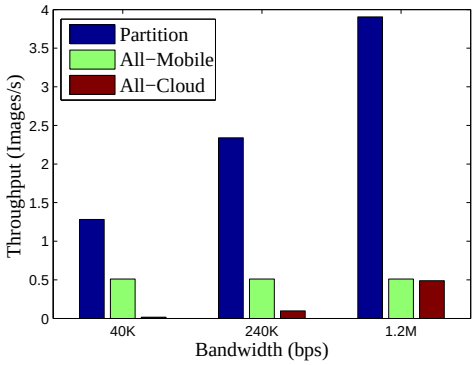


Fig. 4.8: QR-Code recognition performance

Chapter 5

Computation Repartitioning in Dynamic Mobile Cloud Environments

In this chapter, we study the computation repartitioning in dynamic mobile cloud environments, where the device and network status can vary during the execution time of the application. This chapter is organized as follows. Section 5.1 gives the overview of this chapter. In Section 5.2, we present a framework for computation repartitioning in dynamic mobile cloud environments. We take the dynamic network connectivity to clouds as a case study, and solve the computation repartitioning problem in Section 5.3. In Section 5.4, the evaluation results are presented. Section 5.5 concludes this chapter.

5.1 Overview

In recent years, the proliferation of sensors on mobile devices enable a couple of new advanced mobile applications such as augmented reality, collaborative learning, multimedia recognition and retrieval, mobile social gaming and so on. The applications often require intensive and continuous processing of the sensory data. Although the hardware's computing capability increases a lot, running the applications on mobile devices still face problems arising from the constraint on computing capability and/or battery of the device.

On the other hand, the ubiquity and increasing bandwidth of wireless access available to mobile users, and the richness of cloud infrastructures, provide opportunities to develop mobile applications using cloud computing technologies. The most efficient technique used to solve the computing constraints on mobile devices is to offload computations from the device side to the cloud side [KL08] [WGKN08] [YOC08] [BKMS13]. By using computation offloading technique, we need to solve the *computation partitioning* problem, which is to partition the application execution between the device side and the cloud side, such that the execution cost such as the latency is minimized. The partition of the application usually depends on the execution environment including the network connection status and the device status.

The computation partitioning problem has been extensively studied in previous research [CBC10] [CIM+11] [ZKJG09] [GRJ+09] [LWX09] [KAH+12] [RSM+11] [BSPO03] [YCY+13]. These works assume the stable/static execution environments during the life cycle of the application, and thus perform one time partitioning according to the execution environment when the application is initiated. The life cycle is defined as the execution time of the application that lasts from the start to termination. However, this assumption does not hold in reality. For instance, the network connectivity can fail when there is no wireless signal or the signal is too weak to maintain a connection. Even when the network is connected, the bandwidth can fluctuate because of user's mobility. Besides the network status, the device status such as the CPU load may vary during the course of the application execution. With the varying network and device status, one time compuation partitioning may yield performance degradation.

In this chapter, we propose the technique of *computation repartitioning* in the dynamic mobile cloud environment, where the network connection to the cloud and device status can vary with time. Computation repartitioning updates the partition of application from time

to time during its life cycle, according to the estimation and/or the prediction of parameters of the execution environment including the network connection status and device status. More specifically, we first design a framework for run time computation repartitioning in dynamic mobile cloud environments. The framework provides models and mechanisms to conduct the repartitioning of application at the run time. Based on this framework, we conduct a case study for the computation repartitioning, where the network connection to the cloud fluctuate frequently while the device status is relatively stable. In the case study, we develop an online computation repartitioning method, named as Foreseer. It exploits the knowledge of user's mobility pattern to predict the network status, and then updates the partitioning based on the prediction. We evaluate Foreseer using the data traces that are collected from our campus WiFi testbed. We compare Foreseer against the approach in CloneCloud [CIM+11]. It is shown that Foreseer has 35% better performance in term of the completion time. In the case of more frequent network fluctuations, e.g. walking faster in the campus WiFi environment, Foreseer can perform much better than CloneCloud.

In summary, our contributions in this chapter are three folds. First, to the best of our knowledge, we are the first to design a framework for run time computation repartitioning in dynamic mobile cloud environments. The framework provides models and mechanisms to solve the performance degradation issue arising from dynamic network and device status. Second, as a case study, we develop an online method, Foreseer, to solve the computation repartitioning problem under the network status fluctuation. We evaluate our method based on real world data traces from our campus WiFi testbed. The result shows that our method can reduce the application completion time by 35% compared with previous approaches.

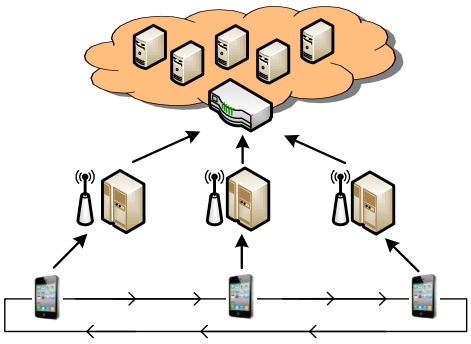


Fig. 5.1: Architectural model of mobile cloud systems

5.2 Terminologies and Application Repartitioning Framework

In this section, we describe the terminologies and our proposed framework for run time computation repartitioning in dynamic mobile cloud environments.

5.2.1 Terminologies

• *Partition*, *Optimal Partition* and *Compuation Partitioning*. Fig.5.1 shows the architectural model for the mobile cloud system. It contains three parts, mobile devices, wireless access network and clouds. The mobile devices can offload some computations of the application to the cloud. Obviously, offloading can reduce the computational cost (e.g., execution time or energy consumption) on the mobile device. Meanwhile, offloading causes additional overhead in data transmissions that are required by the remote execution on clouds. Therefore, in order to minimize the execution cost such as the overall executions latency, it is critical to solve the offloading problem, i.e., to decide whether the application should be offloaded to the cloud or not. For some complex applications which can be divided into a set of dependable parts, we need to make offloading decisions for every part of the application. Note that the decisions for each part are dependent with each other. We name the offloading decisions for all the parts as a *partition* of application. The partition that leads

to the minimum execution cost is named as *optimal partition*. The optimization of compuation partitioning according to the network and device status is named as *compuation partitioning*. Computation partitioning changes the execution model of mobile applications, from single machine execution on the mobile device to distributed execution over the device and the cloud.

Now we are interested in the following question: what factors may affect the optimal partition depend on. First, a good partition must take into account the device status such as the execution speed of the device and the workload on the device. For example, if the computing speed of the device is extremely slow or the workload on the device is high, it is better to offload more computations onto the cloud. Second, a good partition should depend on the network bandwidth, which decides the cost in data transmission for remote execution. Third, the optimal partition depends on the properties of the application itself. The application properties are static, while the device and network status usually change with time.

• *Application Life Cycle* and *Run Time Computation Repartitioning*. *Application life cycle* is defined as the period that the execution of application spans. We also name the period as *run time*. In previous works on compuation partitioning, when the application starts, an optimal partition of the application is determined based on the network and device status at the start time. The partition remains the same during the whole life cycle. *Run time computation repartitioning* is defined as periodically updating of partition of application during its life cycle, based on the changing network and device status, with the aim to reduce the execution cost. Fig.5.2 illustrates the difference between compuation partitioning and computation repartitioning. The strip with various color represents different partitions. The length of strip indicates the execution time. In compuation partitioning, the application execution sticks to one partition during its life cycle, while in computation

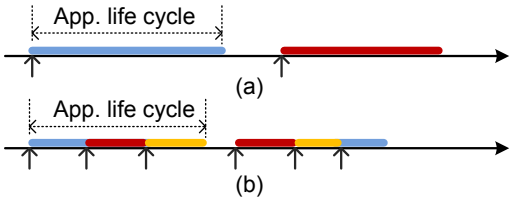


Fig. 5.2: Illustration: a) computation partitioning; b) computation repartitioning

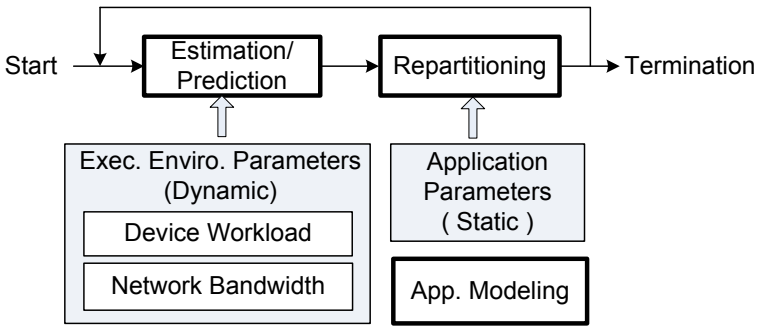


Fig. 5.3: Functional components for computation repartitioning

repartitioning, the application runs with different partitions.

5.2.2 Computation Repartitioning Framework

We now describe the computation repartitioning framework as shown in Fig.5.3. When the the application is initiated, the prediction component is activated to predict the execution environment parameters, including the device workload and network bandwidth. The repartitioning component outputs a partition based on the predicted environment parameters, and its current execution state. The prediction and repartitioning is alternatively performed during the application life cycle, until the application terminates. The objective of periodical repartitioning is to minimize the total execution cost during the application's life cycle. Our framework provides methodologies for application modeling, dynamic mobile cloud environment abstraction, and formulation of the repartitioning procedure. We describe the details in the following.

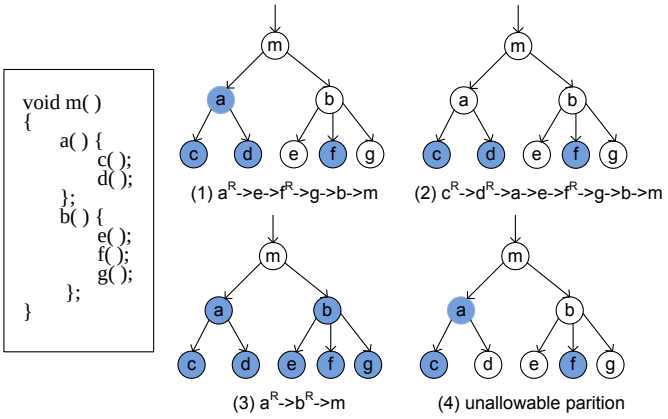


Fig. 5.4: Program tree, legal partitions and the corresponding execution order.

Application Modeling

In our framework, we focus on the method level partitioning of an application. We apply the virtual machine migration to realize the remote execution of application[CIM+11] [GJM+12]. Each method can be migrated to the cloud. During the migration procedure, the system first captures the runtime state at the mobile device, and then transfers the state to the cloud, and finally reintegrates it back after the execution is finished at the cloud. For simplicity, migration points and reintegration points are restricted to the entry and exit of a method.

We use the method call tree to model the application. In this chapter, we interchangeably use the two names, *method call tree* and *program tree*. Fig.5.4 illustrates an example of the program tree. The tree node represents the method and the edge represents the method invocation. For instance, the edge (i, j) indicates that the method i calls the method j. Suppose the methods are executed sequentially between the device and the cloud. Without loss of generality, we require that the program tree is constructed in a way such that the nodes are executed according to the *post-order traversal*. Note that *nested migration* is forbidden. It means that if one node is migrated onto cloud, all its child nodes should be

executed at the cloud. Fig.5.4(1)(2)(3) shows the legal partitions and the corresponding execution order of the methods, where Fig.5.4(4) shows an unallowable partition, because when the method 'a' is migrated to the cloud, all the children nodes of 'a' should be executed remotely since they are forbidden to migrate back to the mobile side. Note that the colored node means remote execution in cloud, while the others are executed locally. In the execution order, a^R means method a is executed remotely.

We define the following variables for a program tree.

• $C_m(i)$ - The execution cost of method i on the device.

• $C_c(i)$ - The execution cost of method i on the cloud.

• $C'_m(i)$ - The *residual cost* of method i. Normally the cost of the parent node i is larger than the summation of the costs of all its child nodes, because node i contains the cost of running the body of code excluding the costs of the methods called by it. We define *residual cost* for each non-leaf node i by $C'_m(i) = C_m(i) - \sum_{j \in ChildOf(i)} C_m(j)$.

• $S_u(i)$ - The size of VM state that needs to be transmitted to the cloud when the method i is migrated onto the cloud.

• $S_d(i)$ - The size of VM state that needs to be transmitted back to the device when the method i is re-integrated from the cloud to the device.

• $C_s(i)$ - The migration cost of method i. Fig.5.5 shows that the whole migration procedure contains five phases: suspension, state transfer (uplink), remote computation, state transfer (downlink) and resuming. We assume that the suspension cost C_{susp} and resuming cost C_{resm} are constants for all the methods.

Given the variables above, we can easily obtain the optimal partition, i.e., to decide which methods are executed on the device, and which methods are migrated to the cloud, such that the total execution cost is minimized. The *snapshot partitioning problem* has been well formulated and solved in [CBC10] [CIM+11]. In practical systems, the execution

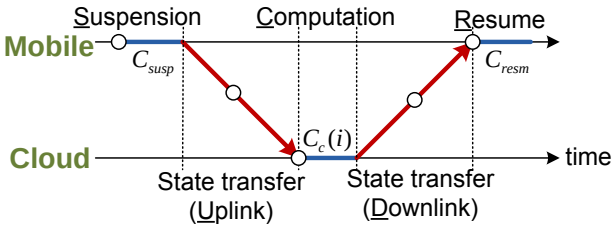


Fig. 5.5: Execution cost in migration

cost $C_m(i)$ and migration cost $C_s(i)$ can change during the life cycle of the application in dynamic mobile cloud environments. We need to update the partition accordingly during the application life cycle such that the total execution cost over the life cycle is minimized.

Dynamic Environment Modeling

We present a simple model of the dynamic mobile cloud environment which includes the device status and network status. The device status affects the execution cost on devices $C_m(i)$. In specific, $C_m(i)$ depends on the computing capability and workload of the device. It is formulated by

$$C_m(i) \propto P \times (1 - \eta),\ 0 \leq \eta < 1, \tag{5.1}$$

where P denotes the computing capability, i.e., the speed that the device processes the program. η denotes the normalized workloads on the device. It represents the percentages of CPU that have been occupied, where $\eta = 0$ indicates that the CPU is totally idle. The computing capability P is static while the workload η is dynamic.

The network status affects the migration cost $C_s(i)$. In specific, $C_s(i)$ depends on the bandwidth of the network connection to the cloud. If we denote the uplink bandwidth by B_{ul}, and the downlink bandwidth by B_{dl}, $C_s(i)$ is calculated by

$$C_s(i) = S_u(i)/B_{ul} + S_d(i)/B_{dl} + C_c(i) + C_{susp} + C_{resm} \tag{5.2}$$

In summary, the dynamic parameters of mobile cloud environment that we consider in

this chapter are device workload η, and network bandwidth (B_{ul}, B_{dl}). In our framework, to guarantee the accuracy, we need to periodically update the estimation and/or prediction of the parameters.

Formulation of Compuation Repartitioning

We formulate the repartitioning as a stochastic dynamic decision process. The decision is made every time the execution environment parameters are predicted. We use $n = \{0, 1, 2, ..., N\}$ to denote the decision epoch. The whole life cycle of the application is divided into N decision epoches. We use T_n to represent the length of epoch n. $n = 0$ represents the initial time when the application is launched. The decisions are made at the beginning of every epoch.

System state: The system state is characterized by the joint knowledge of the program *execution status* $\overline{x}_n$ and the *execution environment status* $\overline{y}_n$, i.e., the device workload η and the network bandwidth (B_{ul}, B_{dl}). At each decision epoch n, *execution status* $\overline{x}_n$ reflects the snapshot information of the program. It contains three parameters $\overline{x}_n = (m_n, p_n, \overline{q}_n)$: 1) the method m_n that the program is running at, 2) a binary variable p_n indicating whether it is migrated or not, $p_n = 1$ if the method is being in migration, otherwise $p_n = 0$, and 3) $\overline{q}_n$ indicating how much computation has been done for this method. If the method runs locally, q_n is assigned with the time that the method has been executed; otherwise if the method is in migration, $\overline{q}_n$ is assigned with: i) the phase that the migration is in, as shown in Fig.5.5; ii) and the corresponding data size that has been sent to or received from the cloud if the migration is in the phase of state transfer.

We treat $\overline{z}_n = (\overline{x}_n, \overline{y}_n, T_n)$ as the system state at the decision epoch n. We add another state $\overline{z}_n = \mathcal{T}$ to denote the application has been finished. The decision process is terminated when the system enters into state $\mathcal{T}$.

Decision: At each epoch n, after observing the system state $\overline{z}_n$, a decision $\mu_n(\overline{z}_n)$ has

to be made. The decision at each epoch n contains: 1) whether or not to terminate the migration procedure if the application is in state of migration; 2)updating the migration points for the methods to be started.

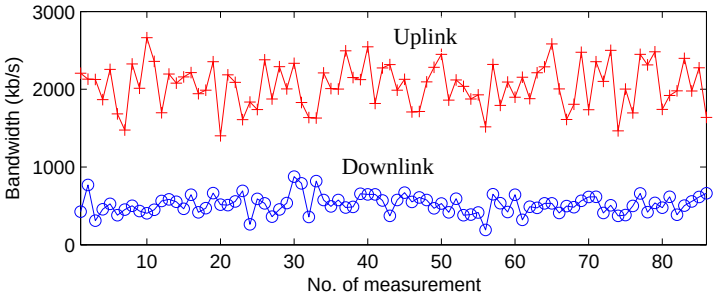
Cost Functions and Optimal Partitioning Policy: The objective of the problem is to minimize the execution time of the application. We define the cost function at each epoch $g(\overline{z}_n) = T_n$ if the program is not completed $\overline{z}_n \neq \mathcal{T}$; otherwise $g(\overline{z}_n) = 0$. The objective function is given by $\rho = \sum_{n=0}^{N} g(\overline{z}_n)$. The sequence of decisions $\phi = \{\mu_0(\overline{z}_0)$, $\mu_1(\overline{z}_1)$, ..., $\mu_N(\overline{z}_N)\}$ is considered as the policy for the dynamic decision process. Let ρ_ϕ be the execution time of the application under the partitioning policy ϕ. The optimal partitioning policy ϕ^* is obtained by $\phi^* = \text{argmin}_\phi \rho_\phi$.

5.3 Case Study: Compuation Repartitioning under Network Bandwidth Fluctuations

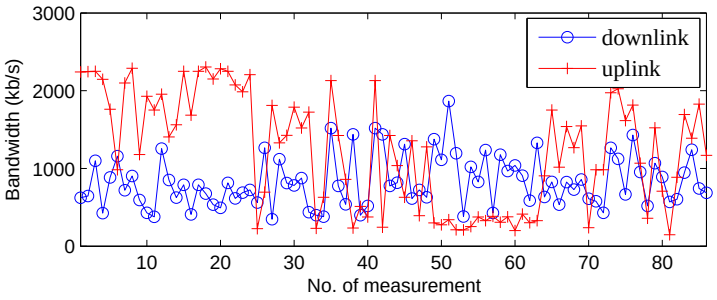
In this section, as a case study, we consider the compuation repartitioning problem under the scenario where the network status encounters connectivity losses and bandwidth fluctuation while the device status is relatively stable.

5.3.1 Network Measurements

The network parameter we are concerned about is the user's bandwidth, because it affects the migration cost of the program. Throughout the chapter, we exchangeably use the terminologies *bandwidth*, *throughput*, *data transfer rate* and *network status*. We consider a wireless network where users access the network through Base Stations (BSs)/Access Points (APs), and roam from one BS/AP to another to remain connected while they move. There exist some places where the wireless signal is too weak to maintain a connection with the AP, or where there is no signal, because this place is not covered by the BSs/APs or the signal is blocked by surrounding obstacles. We define these places as '*holes*'. Inside holes



(a) Stationary Scenario

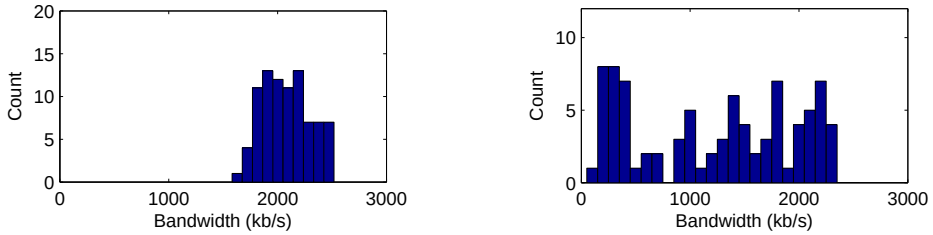


(b) Mobile Scenario

Fig. 5.6: Network bandwidth fluctuation in temporal and spatial domain

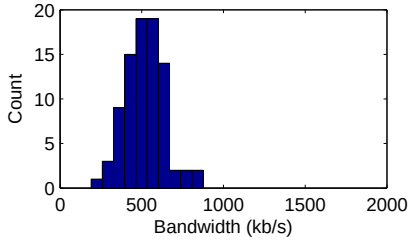
the user encounters connectivity loss. Outside the holes, the bandwidth can vary as the user moves.

To learn how the network status fluctuates in temporal and spatial domain, we compare the network fluctuation between the stationary scenario and mobile scenario. The measurements are done in our campus WIFI testbed with 23 APs deployed. Both scenarios have 86 times of measurements. The measurements are recorded every 20 seconds. Note that in mobile scenario we intentionally avoid the network 'holes'. Fig.5.6 shows that in stationary scenario the network status is relatively stable with time, while in spatial domain networks become more fluctuant as the user's location changes. The distribution histograms of the measurements shown in Fig.5.7 as well illustrates the difference. It is shown that the variance of spatial fluctuation is much larger than that of temporal fluctuation. From above measurements, we make an abstraction that the dominating factor that affects the

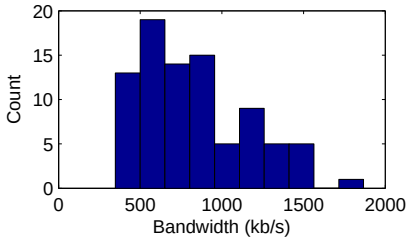


(a) Uplink (stationary)

(b) Uplink (mobile)



(c) Downlink (stationary)



(d) Downlink (mobile)

Fig. 5.7: Histogram of network bandwidth distribution respectively in stationary and mobile scenarios

user's bandwidth is the user's geographical location. The reason is that the life cycle of an application usually lasts a few seconds to several minutes at most. If we look at this short period, the bandwidth fluctuation for one user is dominated by the user's mobility.

5.3.2 Overview of Solution

In Foreseer, we exploit the historical knowledge about the user's mobility to predict the network status. The partitioning of application is then updated based on the predicted network status. In particular, we solve the following problems in the design of Foreseer.

At first, we find that it is extremely hard to accurately measure the network bandwidth in real time through single mobile device. One common approach to measure the bandwidth is by uploading or downloading a large file to or from the server. The uplink or downlink bandwidth will be the size of the file divided by the time used for the uploading or downloading. This measurement itself takes several seconds at least, which is not acceptable for our case. In addition, the high overhead that is incurred during the measurement, e.g.,

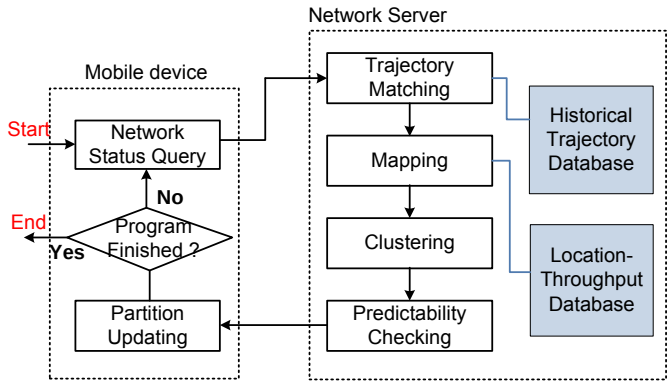


Fig. 5.8: Flow chart of Foreseer

additional data transmission and battery consumption, makes it infeasible to conduct such bandwidth measurement via single mobile device. In Foreseer, we leverage the crowdsourcing to collect the users' bandwidth together with their locations, and learn the probabilistic model of network bandwidth conditioned on the location. The user can query the network bandwidth with its location.

Second, we need to accurately predict the future network bandwidth from the user's mobility. In Foreseer, we deploy a centralized server on the mobile network, named as *network server* throughout the chapter, for: (1) providing the service that allows the mobile users to share location-bandwidth pairs; (2) collecting the users' *historical trajectories*; (3) performing online trajectory matching and network status prediction. To improve the prediction accuracy, we only predict the network status up to certain point in the future, which we name as *predictable duration*. The uncertainty of the network status is relatively low in the predictable duration, while beyond the predictable duration the network status becomes highly uncertain. The network status for one particular user can be predicted based on either the historical trajectories by individuals, or the historical trajectories by all the users. We realize that a centralized server may become the bottleneck of the system when the users scale up. The challenge of designing scalable architectures is inherent to all distributed systems and beyond the scope of this chapter.

With the predicted network status, the core of our problem is to develop an online decision policy for the dynamic repartitioning process. We design an online algorithm that can work efficiently in real systems. Our solution is to make the decision that maximizes the *execution progress* in the predictable duration. Execution progress is defined as the position of the program counter while the application is in execution. In the following, we present a high level description of the protocol which shows how the components work together in our system. Fig.5.8 shows the detailed flow chart of Foreseer.

1. Network Status Query. When the mobile user enters the network, it receives the beacons from the surrounding APs periodically. It measures the signal strength of the beacons, and labels its location with a sequence of AP IDs, which are sorted in a descending order of their signal strength. The location label is null when the user is in 'holes'. When the application is launched, the mobile user sends a series of recorded locations to the *network server*, and queries for its network status.

2. Network Status Prediction. With the series of locations, the *network server* searches the historical trajectories from the database whose prefixes match with the query. All the matched trajectories are mapped into sequences of network bandwidth values, based on the location-bandwidth fingerprint database. Usually, the longer future we attempt to predict, the more uncertain the network bandwidth will be. The network bandwidth sequences are cut off at some time in future, called as *predictable duration*, such that the uncertainty about the network status is low enough. The most likely sequence is returned as the result.

3. Partition Initializing/Updating. Upon receiving the network status, the user determines/updates the partitioning decision such that the execution progress is maximized in this predictable duration. Towards the end of the predictable duration, the mobile user will send the network status query again for the next partition updating.

The three steps are performed iteratively until the program is finished. If the user

happens to be in 'holes' when it sends the network status query, the query will be postponed until the user moves out of the 'holes'. The application sticks on the previous partition.

5.3.3 Network Status Prediction

We describe the details on network status prediction in the following.

Trajectory Representation

We represent the user location with a set of AP IDs from which the mobile user is able to detect the beacon signal. The APs are sorted by a descending order of the signal strength. For example, a legal representation of location is 'abc', in which case the signal from AP 'a' is the strongest, 'b' is less strong and 'c' is the weakest. If the user is located at the holes, the location is labeled as 'N'. There are two cases that the hole 'N' is labeled. The first case is that no AP signal is detected by the user. Second, the signal from the detected AP is too weak to maintain the connection. We have done experiments to learn the threshold of the signal strength, below which the network connection is not possible. We found a threshold of 90 db as an empirical value. The trajectory is represented as a time series of locations. One example of a trajectory is: "ab ab abc bc c c d d N N N f". In the string, we use the space to isolate two locations.

Trajectory Matching

We do not have any requirement on the length of the recorded trajectory. For convenience of description, we can abstract all the trajectories at the database as one virtual string, by joining them together with special isolated symbols. We name the virtual string as *historical string* or *history*, and the string that the user sends for querying the network status as *contextual string* or *context*. The trajectory matching is to find the positions at which the context occurs in the historical string. The substrings of the history, which occur immediately after the context, will be returned as the possible trajectories that the user

will pass by.

In a practical solution, we need to determine a proper length of the context in the string matching. If the context is too long, we may not have enough samples to estimate the probabilistic distribution of the future trajectory. Oppositely, short contexts can not fully capture the feature of the user's input trajectory. In our system, we use the approach in Sampled Pattern Matching (SPM) algorithm [JSA00] to determine the length of the context. In contrast to k-order Markov predictor, which uses a fixed length of the context, the length of context in SPM is decided by a fixed fraction α of the longest context that occurs in the history, where $0 < \alpha < 1$. The method is appropriate to be used in the prediction based on large diverse data sources (in our case, the trajectory traces are from various users at different times).

Speed Alignment

We note that two recorded trajectories are not necessarily identical for the same spatial trajectory. That is because of the diversity of the user's moving speed. For example, the two strings "a a bc bc N N N cd cd" and "a bc N N cd" are likely to be from the same spatial trajectory. Since location sampling frequency has the same influence with speed, we do not separately discuss the case that the frequency is different for users. We onwards assume that all the trajectories have the same sampling frequency.

To estimate the speed difference between the context and history, we generate new trajectories by only recording the location changes. For example, given the original string "ab ab bc bc bc b b N N N c c c", the new string "ab bc b N c" is generated. For convenience of description, we name this operation as *differentiation* which is denoted as $\mathcal{N}(\cdot)$, and use the terms *original string* and *differential string* to distinguish the two strings before and after the operation. The differential string does not contain any speed information. Consider the case that the location sampling frequency is large enough, if two uses move along the

same actual spatial trajectory, their recorded strings should be the same. In our system, the location is sampled once every two seconds. The sampling frequency is high enough to capture all the changes of sensed APs/BSs set along the trajectory, either when the user walks in the WiFi environment or when the user commutes in the cellular network.

We match the differential strings, rather than their original strings, between the context and history. After we find the matched sequence, we compare the length of time that the matched sequence spans in the two original strings, which indicates the speed ratio of the two trajectories. Fig.5.9 illustrates how to calculate the speed ratio of the strings A and B. We first obtain their differential strings $\mathcal{N}(A)$ = "d dc c N ab a" and $\mathcal{N}(B)$ = "b bc c N ab a f fe". After finding that the matched sequence is "c N ab a", we get back to the two original strings and respectively count the duration of the matched sequence. In this example, we can see that A takes 5 time slots to finish the matched trajectory, while B takes 8 time slots. The speed ratio of A and B is $8:5$.

We estimate the speed ratio for each substring of the history that matches with the context. The next string to be predicted after the context is selected from the history based on the speed ratio. For example, if we consider A is the context and B is the history, then the following sequence after the context will be the next $[2 \times \frac{8}{5}], [3 \times \frac{8}{5}], [4 \times \frac{8}{5}], ...$ element after the matched sequence, which is "f f fe ...", where $[X]$ represents rounding X to the nearest integer. Note that we choose the element that changes most recently as the origin point, and count the positions of future elements from the origin point. For example, the origin point is the first 'a' in Fig.5.9. The first element to be predicated has the position 2 in the context, and correspondingly the position $2 \times \frac{8}{5} = 3$ in the history. The 3-rd element in the history from the origin point, e.g., 'f', is chosen as the first predicated element.

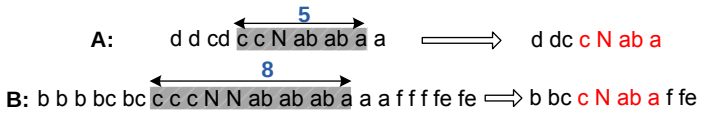


Fig. 5.9: Examples for speed alignment

Mapping

In our system, we build a location-throughput database on the network server. There exists many efficient ways to construct the fingerprint database[HPALP09]. In our system, we use the most intuitive yet efficient method. The database is directly indexed using the location (of string type). For each indexed location, we determine the corresponding throughput simply by averaging of the samples at that location. The throughput for each indexed location is updated as new samples are added. Based on the database, we can map all the possible trajectories into sequences of network throughput values. Note that the location labels 'N' for the network holes are directly mapped into zeros.

Clustering

Suppose that we have M sequences and the length of each sequence is N. Let $\bar{b}_i = (b_{i,1}, b_{i,2}, ..., b_{i,N})$, $1 \leq i \leq M$ denote the sequence i. We perform the clustering for the M sequences. The sequences that are similar to each other will be grouped into one cluster. We measure the similarity between two sequences by their Euclidean distance

$$d(\bar{b}_i, \bar{b}_j) = \frac{1}{N}\sqrt{\sum_{k=1}^{N}(b_{i,k} - b_{j,k})^2}. \tag{5.3}$$

If the distance of two sequences is lower than a given threshold, we say that the two sequences are similar to each other. In the clustering, we construct a graph for all the M sequences, in which the nodes represent individual sequences, the edge represents that the two connected nodes are similar to each other. The clustering is to iteratively find the

maximum clique from the graph. A lot of heuristics have been proposed for the Maximum Clique Problem (MCP) [BB99]. We will not describe the details in our chapter. The cluster that has the largest size is selected to represent the predictable result. In our solution, since the sequences within the cluster are highly similar to each other, we randomly choose one from the cluster as the future network throughput sequence.

Predictability Checking

We define *predictability* by the entropy of the network throughput distribution in future time. Suppose we get K clusters after the clustering for the M throughput sequences. Let $\mathcal{C}_k$ denotes the cluster, where $1 \leq k \leq K$, and $S(\mathcal{C}_k)$ denote the size of the cluster, where $\sum_{k=1}^{K} S(\mathcal{C}_k) = M$. The probability that the actual result is from the cluster $\mathcal{C}_k$ is $p(\mathcal{C}_k) = \frac{S(\mathcal{C}_k)}{M}$. With the probabilistic distribution of the clusters, the predictability is defined by the entropy $H = -p(\mathcal{C}_k) \log_2 p(\mathcal{C}_k)$. Entropy reflects the uncertainty of the future network status. Greater entropy indicates the lower predicability. Generally, the longer the network status to be predicted is, the lower the predictability is.

In our solution, we start from a small prediction duration and check the entropy. If the entropy is lower than a threshold H_{th}, we continue to extend the prediction duration in future. The prediction procedure is terminated until the entropy of the network throughput distribution exceeds H_{th}. We name the length of the period as *predictable duration*. In the computation repartitioning problem, the length of the decision epoch is the predictable duration. Note that we distinguish the uplink and downlink in our system, since our real measurements show the two are quite different. In our solution, we treat them as two independent variables. After we obtain the possible trajectories, we predict the uplink throughput and downlink throughput independently. It is possible that the two have different predictable durations. In this case, we choose the smaller value as the predictable duration, without affecting the predictability of the other one.

5.3.4 Compuation Repartitioning

We first describe the solution for the offline partitioning problem, where the future network status is perfectly known. By using our offline solution, we further design an online algorithm for the compuation repartitioning problem.

Offline Algorithm

The offline problem is to determine an optimal partition of the application, assuming that the network status in the future is known, such that the completion time of the application is minimized. The offline partitioning problem is different with snapshot partitioning problem. In snapshot partitioning problem, the migration cost $C_s(i)$ of each method is constant, while in offline partitioning problem the migration cost depends on when the migration happens. Existing works [CBC10] [CIM+11] aim to solve the snapshot partitioning problem, but can not solve the offline partitioning problem.

Algorithm 2: The offline algorithm

Offline Algorithm: ***FindOptimalPartion***(i, t)
$t_c \leftarrow t$;
if $i \in LeafNodes$ **then**
  $t_{nmgr}(i) \leftarrow C_m(i)$;
else
  $t_{nmgr}(i) \leftarrow C'_m(i)$;
end
while $j \in ChildrenOf(i)$ **do**
  $t_{nmgr}(i) \leftarrow t_{nmgr}(i) + FindOptimalPartion(j, t_c)$;
  $t_c \leftarrow t_c + FindOptimalPartion(j, t_c)$;
end
if $t_{nmgr}(i) > C_s(i, t_0)$ **then**
  $Y(i) \leftarrow 1$; // Method i is migrated into cloud;
  $t_{opt}(i) \leftarrow C_s(i, t_0)$;
else
  $Y(i) \leftarrow 0$; //Method i is executed locally;
  $t_{opt}(i) \leftarrow t_{nmgr}(i)$;
end
return $t_{opt}(i)$;

We develop a recursive algorithm for the offline partitioning problem as shown in Algorithm 2. The input variables includes the local execution cost of each node $C_m(i)$, and the

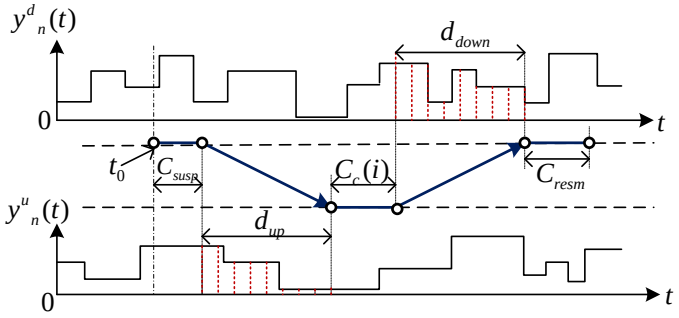


Fig. 5.10: How to calculate migration cost

migration cost related variables such as C_{susp}, C_{resm}, $C_c(i)$, $S_u(i)$, $S_d(i)$, and the dynamic network uplink and downlink bandwidth, denoted as $B_{ul}(t)$ and $B_{dl}(t)$. Note that the network bandwidth during the time period $(0, \infty)$ is known. The application is launched at some time point t. The algorithm outputs the optimal partition. Let $Y(i)$ represent the partition. $Y(i) = 1$ if node i is migrated onto the cloud, otherwise $Y(i) = 0$. Algorithm 2 can optimize the partition for any program subtree i that is going to start at time t, and return the corresponding completion time. Since the network bandwidth is changing over time, the migration cost of node i depends on the time it is started. In the algorithm, $C_s(i, t_0)$ represents the migration cost of node i if the migration procedure starts at time t_0. Fig.5.10 shows the migration cost as given by $C_s(i, t_0) = C_{susp} + d_{up} + C_c(i) + d_{down} + C_{resm}$, where d_{up} and d_{down} are the state transfer time that satisfy:

$$\int_{t_0+C_{susp}}^{t_0+C_{susp}+d_{up}} y^u(t)dt = S_u(i), \tag{5.4}$$

$$\int_{t_0+C_{susp}+d_{up}+C_c(i)}^{t_0+C_{susp}+d_{up}+C_c(i)+d_{down}} y^d(t)dt = S_d(i). \tag{5.5}$$

Online Algorithm

We first define application *execution progress* and then describe the online algorithm. *Execution progress* is defined as the position of the program counter while the application is in execution. In particular, if the program is being in migration, execution progress is defined as the position of the migration point, although it is possible that the actual program counter at the cloud side is in advance of the migration point. Note that *execution progress* is different with the *execution status* defined in Section 5.2. Fig.5.11 illustrate execution progress given the application *execution status*. Suppose that the application is totally completed at mobile device, we can construct a *progress bar* that shows the start time and end time of each method. In this example, the progress bar is in the order of {1, 2, ..., 8}. Note that the length of execution time of the non-leaf node is its *residual cost* $C'_m(i)$, which represents the cost of running the body of code excluding the costs of the methods called by it. In our solution, we simplify that the body of code of the non-leaf node is executed after all its children nodes. After the progress bar is constructed, for any feasible application execution status, we can shows its progress on the bar. The left tree shows that method '7' is being in migration. The execution progress is the migration point of '7', which is right after the completion time of method '3'. The tree in the middle shows that method '7' is being executed on the mobile device. The progress is located at some position of '7' on the bar. In the right tree, method '5' is being in migration, thus the execution progress is at the end of '4' on the bar.

In the online solution, we maximize the application *execution progress* at each decision epoch. According to the definition of execution progress above, if and only if the method in migration can be returned before the end of the current epoch, it will contribute to the execution progress. Oppositely if the method is migrated but not re-integrated in this epoch, the time that is spent on the method migration has no contribution to the execution

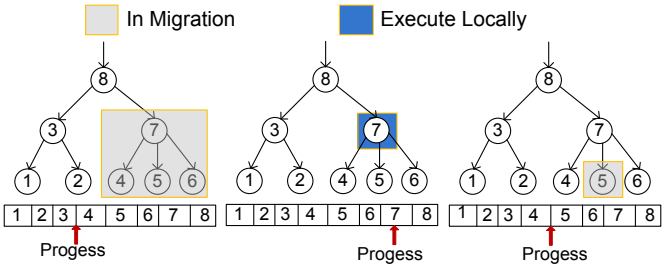


Fig. 5.11: Execution progress

progress. This is because of our pessimistic estimation about the network status beyond the predictable duration. The worst case that the communication is fully disconnected would happen beyond the epoch. Thus, we have a conservative migration policy in our online solution: if the method is able to re-integrate back to the device in the epoch, the migration of the method is allowed; otherwise the migration is not allowed, because in this case we can always obtain more progress by executing the method locally. We conclude that the problem of finding the partition that maximizes execution progress at current decision epoch is equivalent to the problem of finding the partition, that minimizes the completion time of the application, given that the network throughput beyond the predictable duration is zero.

Algorithm 3 shows the online algorithm of the compuation repartitioning problem. The algorithm first checks the execution state $\overline{x}_n = (m_n, p_n, \overline{q}_n)$. If the method m_n is being in migration procedure, the algorithm makes the decision X_n of whether to terminate the migration or not, by comparing the time t_{cont} that the subtree m_n needs to be finished if the migration continues, and the time t_{term} if m_n terminates the migration and seeks a different partition. Note that the overhead of switching to different partition is C_{resm}. t_{cont} can be estimated based on the application *execution status* and network status. As the network throughput beyond the this epoch is assigned with zero, if the migration is still not finished in this epoch, we have $t_{cont} = \infty$. Line 13 to 20 is to update the partition for the nodes not

Algorithm 3: The online algorithm

Online Algorithm: $\mu_n(z_n)$

if *The program is being in migration* $p_n = 1$ **then**
  $t_{cont} \leftarrow EstimateFinishTimeOf(m_n)$;
  $t_{term} \leftarrow C_{resm} +$ ***FindOptimalPartition***(m_n, C_{resm});
  if $t_{cont} < t_{term}$ **then**
    $X_n \leftarrow 0$; $t \leftarrow t_{cont}$;
  else
    $X_n \leftarrow 1$; $t \leftarrow t_{term}$;
  end
else
  $t = EstimateFinishTimeOf(m_n)$;
end
$i \leftarrow m_n$;
while $j \leftarrow NextNodeOf(i)$ **do**
  if $j = ParentOf(i)$ **then**
    $t = t + C'_m(j)$;
  else
    FindOptimalPartition(j, t);
  end
  $i \leftarrow j$;
end

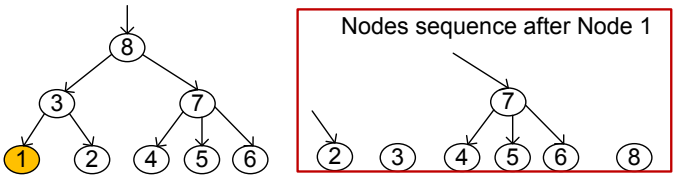


Fig. 5.12: Nodes sequence for $NextNodeOf()$

started. When we search the next node to be started, we always search the subtree that includes as more children that are not started as possible. Fig.5.12, for example, shows the next node after 1 is nodes 2, 3, 7, 8 rather than 2, 3, 4, 5, 6, 7, 8.

The predicted network status is not ideally the same with the reality, so the program is likely to end up at each epoch with a method being in progress of migration. In this case, the decision on whether or not to terminate the migration procedure is necessary. At each epoch, we make the partition for all the nodes to be started rather than the nodes that are likely to be started in current epoch. It causes more overhead but it is reliable to handle the case in which the network throughput happens to be much better than what we predicted,

and some nodes that were not considered to start in the epoch start eventually.

5.4 Evaluation

5.4.1 Evaluation Setup

We collect the network bandwidth traces from our campus WiFi network testbed. We deploy 23 WiFi access points in the test area. Note that in the Fig.5.13 only the nodes labeled with figures are deployed with APs. The 23 APs are densely deployed at the four buildings. Some buildings with APs deployed start from the first floor, so people can freely pass through them on the ground. The maximum length and width of the erea are approximately 500 meters and 700 meters. The APs are mainly deployed at four buildings. Since APs are not intentionally deployed to cover the whole test area, there exists a few network holes both in the buildings and in the open space of the campus. Our prediction method needs the database of historical trajectories. Before collecting the trajectories, we have built a *mobility graph* that constrains the user's mobility due to the environment restrictions. The users' trajectories correspond to pathes in the graph. Fig.5.13 shows the AP deployment and mobility graph in our test area. We have collected data for 30 trajectories for six users. The trajectories are not totally different, but have a few overlaps between each other. The locations are recorded every two seconds along the trajectory, but each trajectory has different speed that ranges from approximately 1.0 m/s to 2.5 m/s. The time length of each trajectory is about 10 minutes.

In order to build a snapshot map of location-throughput, we have selected 200 positions to measure the network bandwidth which covers the whole test area. At each position we have ten measurements of both the downlink and uplink bandwidth. The average on the 10 measurements is recorded as network status at that position. The measurements are collected from 10:00 am to 12:00 am on Mar 2nd, 2013. Note that the positions we

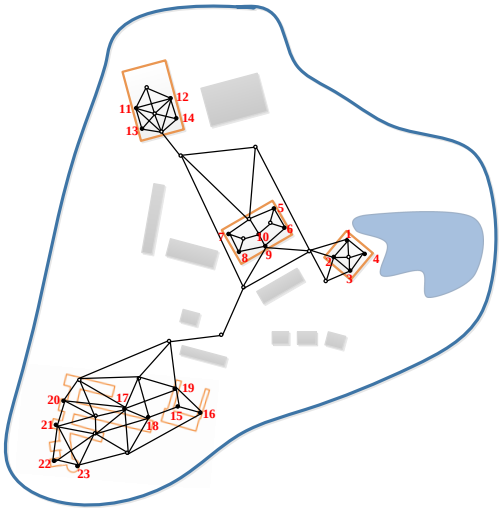


Fig. 5.13: APs deployment and Mobility Graph.

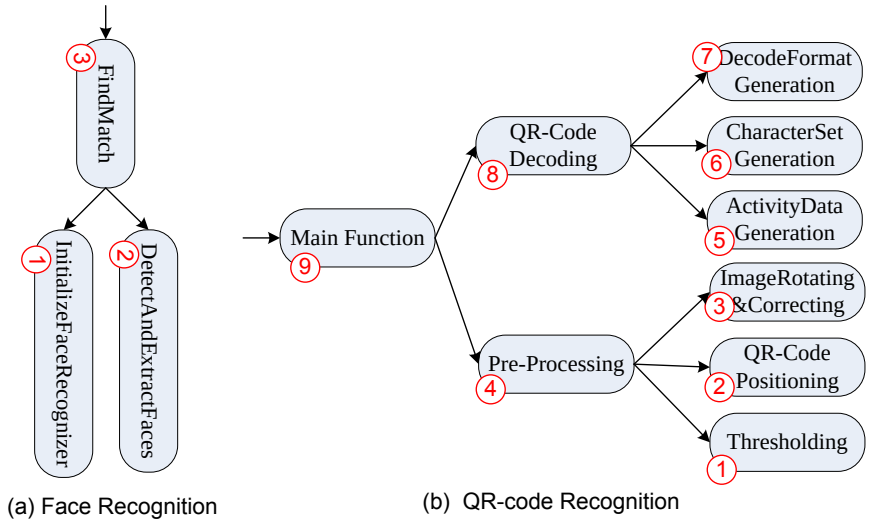


Fig. 5.14: Program trees used in evaluation

selected do not include network holes, because the network bandwidth in holes can be directly mapped into zero. In real systems, the snapshot map of location/throughput varies depending on workloads on APs, traffic on the backbone and so on. It needs to be updated periodically, e.g., one update per hour. In our evaluation, we simply collect the snapshot map once and use it all the time.

We evaluate our repartitioning algorithm using two applications: face recognition and

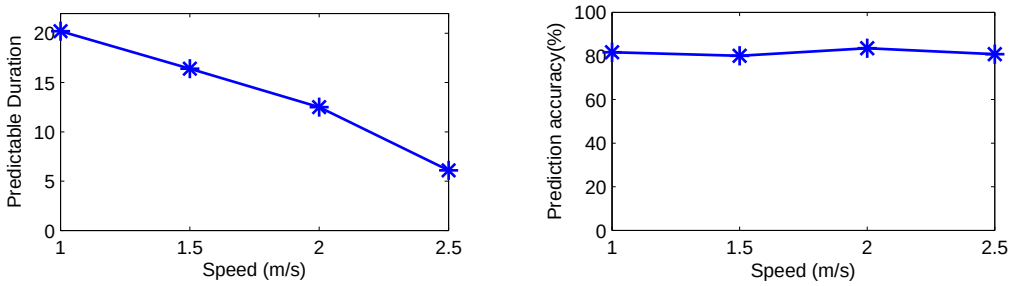
Table 5.1: Application parameters

Application	Methods	$C_m(i)$	$C'_m(i)$	$C_c(i)$	$S_u(i)$	$S_d(i)$
Face	①	7.1 s	7.1 s	0.71 s	2.3MB	2.3MB
Recognition	②	19.6 s	19.6 s	1.96 s	6.9MB	6.9MB
	③	28.1 s	1.4 s	0.14 s	91kB	91kB
QR-Code	①	800 ms	800 ms	80 ms	350KB	87.5KB
Recognition	②	1300 ms	1300 ms	130 ms	87.5KB	62.5KB
	③	110 ms	110 ms	11 ms	62.5KB	87.5KB
	④	2210 ms	0 ms	221 ms	350KB	87.5KB
	⑤	500 ms	500 ms	50 ms	62.5KB	50.3KB
	⑥	400 ms	400 ms	300 ms	62.5KB	50.1KB
	⑦	300 ms	300 ms	30 ms	62.5KB	50.1KB
	⑧	1480 ms	280 ms	148 ms	87.5KB	63KB
	⑨	3690 ms	0 ms	369 ms	350KB	52KB

QR-code recognition. These two applications are also used in most related works [CBC10] [CIM+11] [YCY+13]. Fig.5.14 shows the method level graph of the two applications. The details about the application parameters are shown in Table 5.1, where the data of face recognition are from [CBC10] and the data of QR-code recognition are from [YCY+13]. Note that the computation in cloud is 10 times faster than on mobile devices. The suspension cost C_{susp} and resuming cost C_{resm} are set as 1 second. These are typical values used in [CIM+11].

5.4.2 Network Status Prediction

We first evaluate our approach for network status prediction. We concern on two metrics: *accuracy* and *predictable duration*. The accuracy is measured by the number of successful prediction over the total number of predictions. *Successful prediction* means that the predicted result is similar to the ground truth according to Equation 5.3. The predicted result is obtained through mapping the predicted trajectory into network bandwidth sequence by using the location-bandwidth map. However, to obtain the ground truth, we first map the real trajectory into network status sequence. Considering the stochastic property of



(a) Prediction accuracy

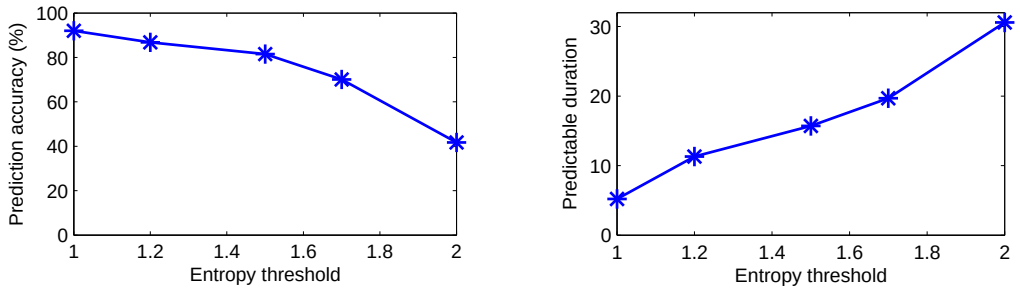
(b) Predictable duration

Fig. 5.15: Performance of network status prediction varies depending on the walking speed of user

bandwidth in temporal domain (shown in Fig.5.6a), the ground truth are then added with a simulated Gaussian noise. The mean and variance of the noise added for uplink and downlink bandwidth are respectively set as $(0, 400kbps)$ and $(0, 100kbps)$.

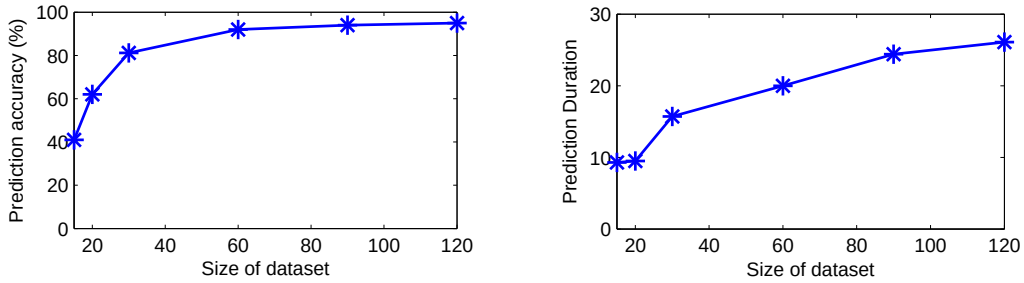
We evaluate the overall performance of the prediction method. We choose one of the 30 trajectories as test trajectory, and the left as the historical trajectories. We simulate the online predictions with the test trajectory. We repeat this evaluation on all 30 trajectories. Finally, we have 1123 times of prediction, where each trajectory has $1123/30 = 37$ times of predictions on average. Among those predictions, 912 predictions are successful, thus the overall accuracy of our method is 81.2%. The average predictable duration is 15.7 seconds. The result shows we can accurately predict the network status in future 15.7 seconds with our prediction method.

Next, we evaluate how the walking speed affects the prediction performance. We classify the trajectories into four categories according to the speed: slow (about 1m/s), medium (about 1.5m/s), fast (about 2.0m/s), and very fast (about 2.5 m/s). For the test trajectories which belong to the same speed category, we count the the total number of online predictions the number of successful predictions, and the average predictable duration. Fig.5.15 shows predictable duration decreases as the speed increases, while the accuracy almost remains the same. The reason that the predictable duration changes is, that the length of future



(a) Prediction accuracy (b) Predictable duration

Fig. 5.16: Performance of network status prediction varies depending on H_{th}



(a) Prediction accuracy (b) Predictable duration

Fig. 5.17: Performance of network status prediction varies depending on the data size

spatial trajectory that we are able to predict is constrained. If the users move fast, the time that user passes by the predictable area will be short. The speed does not affect the accuracy because of the threshold H_{th} that is applied to stop prediction if the network status becomes high uncertain in further future.

We further evaluate the effect of parameter H_{th} on the prediction accuracy. Increasing H_{th} means allowing prediction in uncertain network status distribution. In this case, although the predictable duration could be increased, the prediction is more likely to deviate from the ground truth. Fig.5.16 shows prediction accuracy decreases as we increase the threshold H_{th}. We will report how H_{th} influences the performance of program partitioning in next subsection.

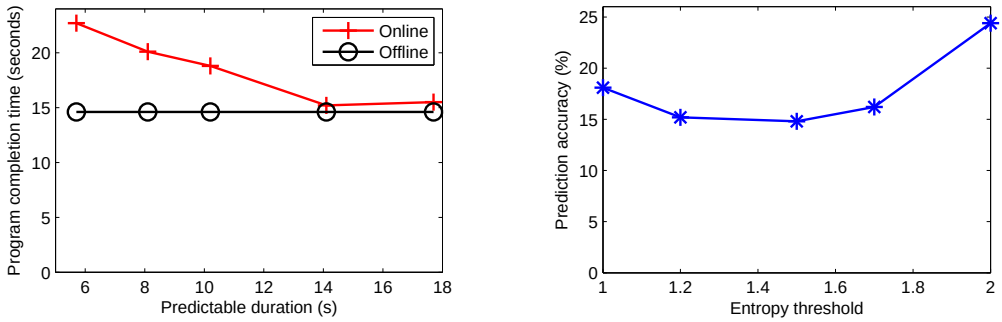
Finally, we explore the impact of the trajectory data size on the prediction performance.

First, we discard some of the 30 trajectories data set. As shown in Fig.5.17, we find both the accuracy and predictable duration decrease significantly. The reason is that the test trajectory may have some spatial intervals that never appear in the historical trajectory database. In this case, it is difficult to find the matched trajectory from the database, or even when matched trajectories are searched from the database, but the successive part of the test trajectory appears to be different from all the matched trajectories. Second, we increase the data size by simulating more trajectories. The new trajectories are generated by randomly picking up the intervals from the 30 real trajectories and joining them together. Fig.5.17 shows that both prediction accuracy and predictable duration increases as the data size increases. In particular, the increase of predictable duration is very obvious. Overall, this evaluation implies that the performance of our prediction method highly relies on the data size. If we want to be able to predict more time in future or more accurately, we should have more samples in the historical database.

5.4.3 Compuation repartitioning

Metric: Completion Time. We compare the online compuation repartitioning method (Algorithm 3) respectively with CloneCloud, which runs the application with one time partitioning based on the current network status when the application is launched, the offline partitioning method (Algorithm 2) and the baseline case that the application is totally executed on the mobile device. The main performance metric is *completion time* of the program. The performance of the offline partitioning algorithm actually reflects the upbound that the online algorithm can achieve. The algorithms are evaluated using the network traces we have collected.

First, we evaluate the overall performance of the partitioning methods. We have collected 30 network traces. For each network trace, we repeatedly run the application 50 times



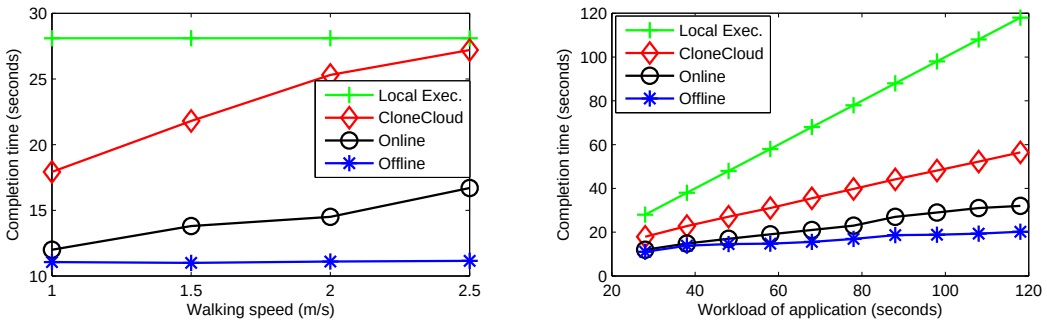
(a) Stationary Scenario (b) Mobile Scenario

Fig. 5.18: The completion time varies depending on: a)the predictable duration; b) H_{th}

by randomly selecting the application launching time along the trajectory. Thus, the application runs $50 \times 30 = 1500$ times for each method. We obtain the average completion time for the online algorithm, CloneCloud, offline algorithm and the baseline method. Fig.5.20a and Fig.5.20b respectively show the program completion time under the four methods for the two applications. For face recognition, Foreseer can reduce the completion time by 41% over CloneCloud. For QR-code recognition, Foreseer can reduce the completion time by 35% over CloneCloud.

We analyze how predictable duration affects the online predictive method. Note that we run the program for 1500 times to obtain the overall performance of the predictive method. Each running of the program requires multiple times of prediction on the network status. We record the average predictable duration as well the program completion time during each running of the program. We analyze the 1500 samples of the predictable duration and program completion time, and find that program completion time decreases as the predictable duration increase, which is shown in Fig.5.18a. *This result implies that if we can predict more time in future about the network status, we will achieve better performance.*

To evaluate how the performance of predictive partitioning algorithm changes depending on the prediction parameter H_{th}, we assign H_{th} with different values, and repeat 1500 runs



(a) walking speed (b) application workload

Fig. 5.19: The completion time varies depending on: a)the walking speed; b) application workload

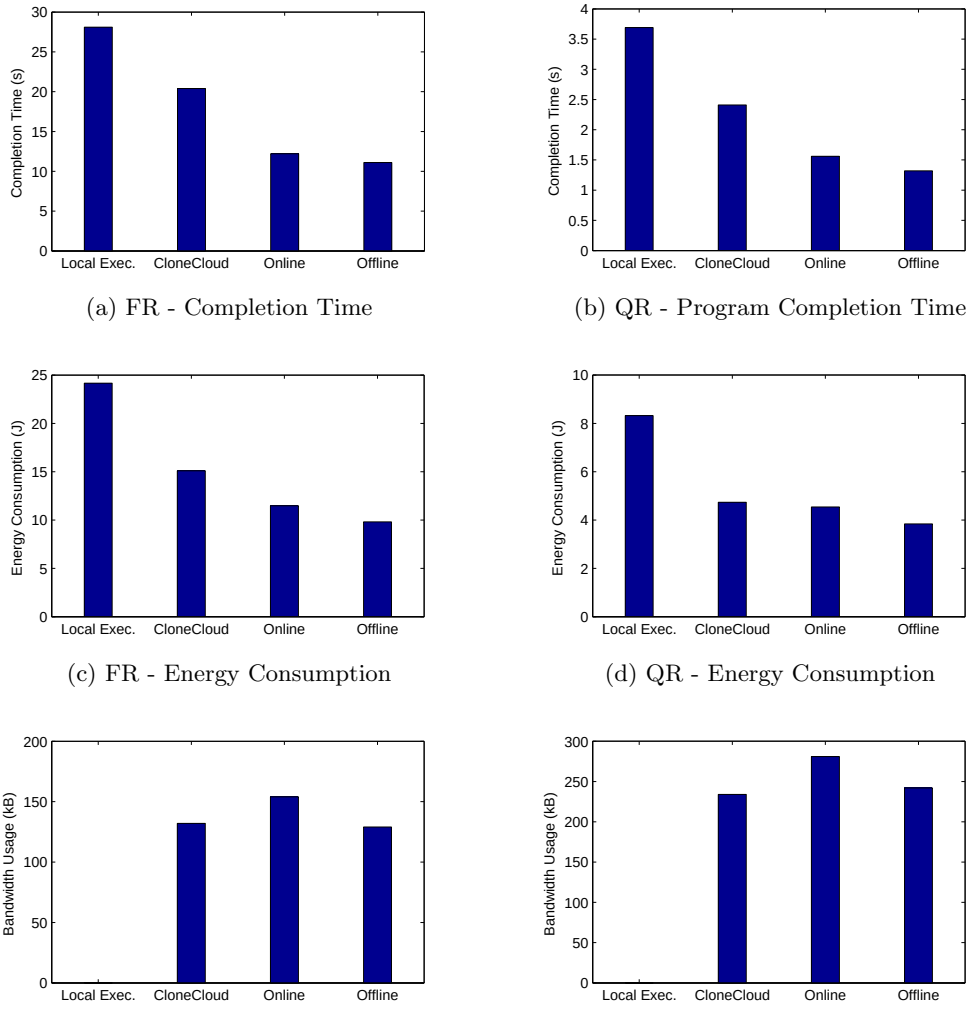
of the algorithm. Fig.5.18b shows that the performance is not good either when H_{th} is too low or H_{th} is too high. The reason is that when H_{th} is too low, although the network status can be predicted accurately (shown in Fig.5.16a), the predictable duration is quite short which degrades the performance. Oppositely when H_{th} is too large, the network prediction is not accurate, which leads to bad performance of the predictive method.

We then evaluate how the walking speed affects the performance of the three partitioning methods. We conduct the test under four speeds 0.5 m/s, 1.0 m/s, 1.5 m/s, and 2.0 m/s. For each speed, we select one trajectory, and also repeat the application 50 times by randomly choosing the application launching time. Fig.5.19a shows the three methods' performance under different walking speeds. The performance of both the online method and CloneCloud degrades as the walking speed increases. The faster the user walks, the more unstable the networks status is, in which case CloneCloud that assumes the stable network has lower performance. The reason that our online algorithm has lower performance as the speed increases is that the predictable duration becomes short when user moves faster. *This evaluation shows that Foreseer always outperforms Clone Cloud, specially when the user moves relatively fast in the network.*

We evaluate the effect of application workload on performance of the partitioning methods. The *workload* is defined as the completion time if the program is totally executed on the mobile device. In this evaluation, we still use the face application, but we simulate large workload by assuming that the same application is executed on a very slow device. Fig.5.19b shows the completion time of the three methods vary depending on the program workload. We can see that the more the program workload is, the better performance our predictive online algorithm has over CloneCloud. *This evaluation implies that Foreseer is suitable to be used in compute-intensive applications, while for small application, it is enough to use current bandwidth to partition the program.*

Other Metrics: Energy Consumption and Bandwidth Usage. Although Foreseer is designed with the objective of minimizing program completion time, we also measure other metrics of Foreseer such as energy consumption and bandwidth usage. The *energy consumption* of Foreseer contains the application execution itself and overhead of network status prediction. To measure the energy consumption of application itself, we use the same energy model with [CIM+11] in our evaluation. For the overhead of Foreseer, we mainly consider the energy consumed on the periodic location sensing. Therefore, we have the following equation to model the energy consumption of Foreseer: $EC = P_{cpu}t_{comp}+P_{net}t_{trans}+E_{sense}$. P_{cpu} and P_{net} are constants. E_{sense} is calculated by the total energy consumed in location sensing among the whole trajectory divided by the run times of the program on this trajectory. The *bandwidth usage* indicates the data amount that are transferred over the wireless networks. This metric is usually concerned by the users who have limited data traffic budget.

Fig.5.20 shows the energy consumption and bandwidth usage for the two applications. All measurements are the average of 1500 runs of the application. We can see that Foreseer outperforms CloneCloud in term of energy consumption for the application of face



(a) FR - Completion Time

(b) QR - Program Completion Time

(c) FR - Energy Consumption

(d) QR - Energy Consumption

(e) FR - Bandwidth Usage

(f) QR - Bandwidth Usage

Fig. 5.20: Performance comparison between four methods: **CloneCloud**, Foreseer (**Online** Algorithm), Foreseer (**Offline** Algorithm) and **Local Execution** without partitioning.

recognition, while for QR-code recognition, Foreseer consumes the same amount of energy with CloneCloud. This is because face recognition has much larger workload then QR-code recognition. Usually Foreseer can save energy through reducing the completion time. *relatively large workload programs can benefit from the Foreseer in term of energy consumption.* The bandwidth usage for CloneCloud and Foreseer(offline) are almost the same for the two applications. However, *Foreseer (online) causes a litter bit higher bandwidth usage than the other two schemes.* This is because the online algorithm of Foreseer can terminate the method migration if it predicts that future network bandwidth is not good. In this sense, the bandwidth used to transmitting data in the terminated migration procedure is a waster.

5.5 Summary

In this chapter, we proposed a framework for run time compuation repartitioning in dynamic mobile environments. Based on the framework, we take the dynamic network connection to the cloud as a case study, and design an online solution for compuation repartitioning under network fluctuations. Our solution exploits the knowledge of user's mobility to predict the future network status. According to the network prediction, we designed the online repartitioning algorithm that aims to maximize the execution progress during current the predictable duration. To evaluate our solution, we collected data set from our campus Wifi testbed that contains the user's walking trajectories and measurements of spatial distribution of network throughput. The evaluation results show that our solution can reduce the completion time of program by at least 35%.

Chapter 6

Multiple User Computation Partitioning

In this chapter, we study the multiple user computation partitioning problem. This chapter is organized as follows. In Section 6.1, we give the overview of this work. In Section 6.2, we present the system model and problem statement. We present our offline algorithm, *SearchAdjust*, in Section 6.3, and a set of benchmark offline solutions in Section 6.4. The online solution for MCPP is presented in Section 6.5. We evaluate both the offline and online solutions in Section 6.6. In Section 6.7, we conclude the chapter.

6.1 Overview

Computation partitioning between the mobile device and the cloud has been studied a lot in mobile cloud computing. Existing work mainly focus on single user computation partitioning, in which the computations are partitioned for one single user without regard to the partitioning results of other users [YCT+12] [CBC10] [CIM+11] [ZKJG09] [GRJ+09] [KL08] [LWX09] [RSM+11]. It is assumed that the cloud always has enough resources to accommodate without delaying the offloaded tasks, no matter how many other users offload the computations on the cloud. However, from the standpoint of the application provider, the assumption is not practical due to the following two reasons. First, the application

providers need to balance the number of resources leased from cloud IaaS providers and the application performance, in order to lower their operational cost [SSSS11]. Second, due to the unpredictable number of mobile users in large scale cloud applications, the application provider can not guarantee all the times to have enough resources to host the mobile users' offloading requests. Therefore, it is necessary to place the computation partitioning problem on constrained number of cloud resources. We name this problem as **Multi-user Computation Partitioning Problem (MCPP)**.

MCPP is much more challenging than the existing computation partitioning problems. In MCPP, the users' partitioning results are dependent with each other because of their competition for the cloud resources. For example, one user's decision on whether to offload the task not only depends on its saved computational cost and communication overhead, but also depends on how many other users offload the tasks onto cloud. The number of users who offload the tasks onto cloud represents the **load** on the cloud. If the load is high, the time spent in waiting for available cloud resources may sacrifice the benefit of offloading. An optimal solution for MCPP requires a unified schedule of all the users' computations onto their mobile devices and cloud resources.

In this chapter, we study the MCPP for latency sensitive mobile cloud application. The problem is to schedule the offloaded computations on a constrained number of cloud resources as well as to partition the computations between mobile side and cloud side for all the users, such that the average application delay is minimized. The selected performance metric is average application delay/latency sine it is the most critical one for latency sensitive mobile cloud applications. Moreover, we study how the application performance changes with the provisioned cloud resources and the load on the system, and thus construct a Performance-Resource-Load (PRL) model. The PRL model provides an optimal tradeoff between the application performance and the cost of cloud resources. We believe the model

can help the application provider to achieve a cost-efficient utilization of the cloud resources, and hence save their operational cost. The main contributions of this work are as follows:

- To the best of our knowledge, this work is the first one to study the Multi-user Computation Partitioning Problem (MCPP), from the standpoint of application providers. The problem jointly considers the partitioning of computations for each user and the scheduling of offloaded computations on the cloud resources. It could help the application provider to achieve optimal application performance when faced with unpredictable number of users.

- We show that our MCPP is different from and more difficult than the existing job scheduling problems, such as Task Scheduling Problem in Heterogeneous Computing (TSPHC) and Hybrid Flow Shop (HFS) scheduling problems.

- We systematically solve the offline MCPP by proposing a set of competitive algorithms. Through the benchmarks, we show that our proposed algorithm, *SearchAdjust*, has better performance than the existing list scheduling algorithms by 10 percents in term of application delay.

- We design an online algorithm for MCPP that can be deployed in practical systems, and demonstrate its effectiveness using real world load traces.

6.2 System model and Problem Formulation

6.2.1 Application model

We target for the *Latency Sensitive Mobile Cloud Applications*, which requires low latency for good user experiences. The applications often take sensory data as input, perform a sequence of operations onto the data, and then output the results. Mobile augmented reality is considered as one typical application. The application uses the camera and/or other

Table 6.1: Mathematical notations in this chapter

j	index of module of the application;
n	total number of modules that the application contains;
c_j	execution time of module j at the cloud side;
w_j	execution time of module j at the mobile device;
π_j	data transmission time between module j and module $j+1$;
x_j	decision variables in SCPP that indicate whether module j is offloaded onto cloud;
λ	number of user's requests;
r	number of cloud servers;
i	index of the user;
k	index of the machine that could be the mobile device or the cloud server;
T	the length of time interval;
δ_i	release time of user i's request;
$w_{i,j}$	execution time of module j on the mobile device of user i;
$\pi_{i,j}$	data transmission time from module j to $j+1$ for user i;
$t_{i,j}$	the completion time of module (i,j);
$x_{i,j,k}$	binary variable that indicates if module (i,j) is executed on server k;
$y_{i,j}$	binary variable that indicates if module j and $j+1$ of user i are executed on different sides k;
$z_{i,i',j,j'}$	binary variable that indicates if the execution of module (i,j) precedes module (i',j');
Δt	the length of time slot;
η	the index of time slot;
λ_η	the number of user's requests at time slot η;
$\overline{\lambda}$	the expectation value of λ_η over all time slots;
var	the variance value of λ_η over all time slots;
r_η	the number of cloud servers allocated to server user's requests at time η;

sensors to perceive the user's environment/scene, and then augment the original scene with relevant information. The perception is done frequently which is driven by the user's input. The core part of augmented reality applications is the image based object recognition. Fig.6.1 shows the operations involved in the whole process of image based object recognition. Note that the SIFT algorithm is used to extract the features [Low04].

In our work, the applications are modeled as a sequence of processing modules (shown as vertex in Fig.6.1). The module represents a kind of operation onto the data. The

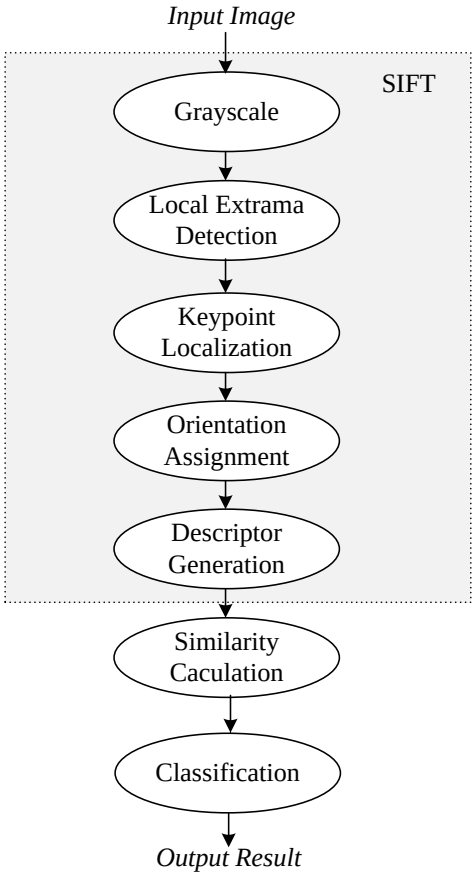


Fig. 6.1: The functional modules of image based object recognition

directed edges represent the dependency between the modules. It means that a module can not start to run until its precedent module completes. Each module is allowed to run either locally on the mobile device or remotely on the cloud. Also the input data of the application is supposed to be from the sensors of the mobile device, and the output data should be delivered back to the mobile device. The performance metric is the execution time of the application. As the execution time represents the responsive delay/latency for the application, we simply use the term **delay** or **latency** in the chapter. The delay is the summation of the computational time of all the modules and the data transmission time between the modules.

6.2.2 Single user computation partitioning

We first describe the *Single user Computation Partitioning Problem(SCPP)*, in which one single user runs the application and requests the cloud for computation offloading. Table 6.1 shows the mathematical notions that we use throughout this chapter. Suppose the application consists of a sequence of n modules. Each module can be executed either at the mobile side or at the cloud side. The execution time of module j is c_j if it is offloaded onto cloud ($1 \leq j \leq n$); otherwise, it is w_j, where $w_j > c_j$. If two adjacent modules j and $j+1$ run on different sides, the data transmission time is π_j; otherwise the data transmission time between j and $j+1$ becomes zero when they run on the same side. To model that the input/output data of the application should be from/to the mobile device, we add two virtual modules 0 and $n+1$ as the entry and exit modules.

Definition 1 *Single user Computation Partitioning Problem(SCPP)*: Given the computation cost c_j and w_j ($1 \leq j \leq n$), and communication cost π_j ($0 \leq j \leq n$), the SCPP is to determine which modules should be offloaded onto cloud such that the application delay

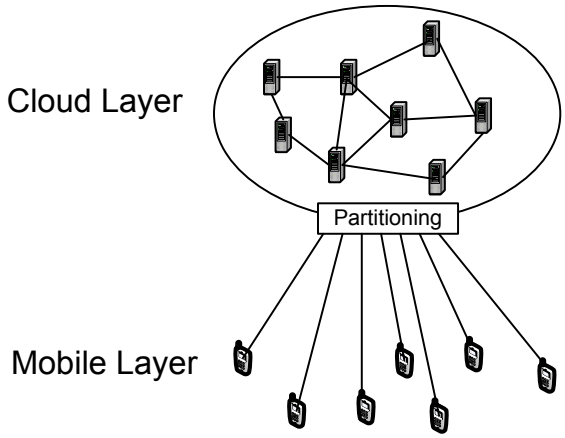


Fig. 6.2: System model of multi-user computation partitioning

is minimized. It is formulated by

$$\min_{x_j} d = \sum_{j=1}^{n}[(1-x_j)w_j + x_j c_j] + \sum_{j=0}^{n} |x_j - x_{j+1}|\, \pi_j, \tag{6.1}$$

where x_j is a binary decision variable. $x_j = 1$ if the module j is offloaded onto cloud, otherwise $x_j = 0$; and $x_0 = x_{n+1} = 0$.

6.2.3 Multiple users computation partitioning

Next, we illustrate the system model of multiple users computation partitioning which is shown in Fig.6.2. The system consists of two layers: cloud layer and mobile layer. At mobile layer, we have a set of users that send requests to cloud for partitioned execution of the application. Upon receiving users' requests, the cloud needs to partition the application for each user, i.e., to decide which modules of the application are executed on the mobile device and which modules are offloaded to the cloud, and also to schedule the offloaded modules onto the cloud servers. In our model, we assume that the users are requesting for partitioned execution of the same application. However, we can extend the model by considering that the users request for various applications, and easily apply the methods in the following sections into the extended model.

We consider a fixed time period $(0, T)$, during which a total number λ of requests are sent to the cloud. Let i denote a particular request and δ_i denote the release time for request i, where $1 \leq i \leq \lambda$ and $0 \leq \delta_i \leq T$. For convenience of description, when we mention user i or i-th user in this chapter, we refer to the user who emits request i.

We model cloud resources as a set of servers/VMs. The number of cloud resources is denoted as r. As mentioned in the SCPP, c_j represents remote computation cost (time) of the j-th module of the application. Usually the mobile users have different processing capabilities and networking bandwidth. Thus, we use a $\lambda \times n$ matrix W to represent local computation cost in which each $w_{i,j}$ gives the execution time to complete the module j on the i-th user's mobile device. Π is a $\lambda \times (n+1)$ communication cost matrix in which each $\pi_{i,j}$ $(0 \leq j \leq n)$ represents the data transmission time from the module j to $j+1$ for the user i.

We first study the offline multi-user partitioning problem, in which we assume perfect knowledge on the requests released from time 0 to T. We develop an offline algorithm as well as a set of competitive benchmark algorithms in Section 6.3 and Section 6.4. Based on the offline solutions, we design an online solution in Section 6.5.

Definition 2 *Multi-user Computation Partitioning Problem(MCPP)*: Given λ, r, δ_i, c_j, $W_{\lambda \times n}$ and $\Pi_{\lambda \times (n+1)}$, the problem is to determine for all the users at which machine (including the mobile device and the cloud servers) and at what time each module is executed, such that the average application delay of the users is minimized.

We formulate MCPP as a Mixed Integer Linear Programming(MILP) problem. For the convenience of description, we use (i, j) to represent the j-th module for the user i. In this formulation, we define one continuous variable $t_{i,j}$ and three 0-1 discrete variables $x_{i,j,k}$, $y_{i,j}$, $z_{i,i',j,j'}$ as follows. $t_{i,j}$ represents the time that the module (i, j) is finished. $x_{i,j,k} = 1$ if the module (i, j) is executed on the machine k, and otherwise $x_{i,j,k} = 0$. Note that

$k = 1, 2, ..., r$ represents the servers at the cloud side, and $k = 0$ represents the mobile device. $y_{i,j} = 1$ if the two dependent modules, (i, j) and $(i, j+1)$, are executed on different sides, and otherwise $y_{i,j} = 0$ if the two modules are on the same side; $z_{i,i',j,j'} = 1$ if the execution of module (i, j) precedes the module (i', j'), and otherwise $z_{i,i',j,j'} = 0$. We also define a positive constant IN which is as great as infinity. The MILP formulation can be written by Equation 6.2.

$$
\begin{aligned}
&\text{Min} \frac{1}{\lambda} \sum_{i=1}^{\lambda} (t_{i,n+1} - t_{i,0});\\
&\text{Subject to:}\\
&\text{(a) } \sum_{k=0}^{r} x_{i,j,k} = 1, \forall i \in [1, \lambda], \forall j \in [0, n+1];\\
&\text{(b) } t_{i,j+1} \geq t_{i,j} + \pi_{i,j} \cdot y_{i,j} + w_{i,j+1} \cdot x_{i,j+1,0} + c_{j+1} \cdot (1 - x_{i,j+1,0}), \forall i \in [1, \lambda], \forall j \in [0, n];\\
&\text{(c) } t_{i',j'} \geq t_{i,j} + c_{j'} - IN \times (1 - z_{i,i',j,j'}) - IN \times (2 - x_{i,j,k} - x_{i',j',k}),\\
&\qquad\qquad \forall i, j, (i, j) \neq (i', j'), \forall k \in [1, r];\\
&\text{(d) } y_{i,j} \geq x_{i,j,0} - x_{i,j+1,0}, \forall i \in [1, \lambda], \forall j \in [0, n];\\
&\text{(e) } y_{i,j} \geq x_{i,j+1,0} - x_{i,j,0}, \forall i \in [1, \lambda], \forall j \in [0, n];\\
&\text{(f) } y_{i,j} \leq x_{i,j,0} + x_{i,j+1,0}, \forall i \in [1, \lambda], \forall j \in [0, n];\\
&\text{(g) } y_{i,j} \leq 2 - x_{i,j,0} - x_{i,j+1,0}, \forall i \in [1, \lambda], \forall j \in [0, n];\\
&\text{(h) } z_{i,i',j,j'} > (t_{i',j'} - t_{i,j})/IN, \forall i, j, (i, j) \neq (i', j');\\
&\text{(i) } z_{i,i',j,j'} \leq 1 + (t_{i',j'} - t_{i,j})/IN, \forall i, j, (i, j) \neq (i', j');\\
&\text{(j) } x_{i,j,k}, y_{i,j}, z_{i,j,i',j'} \in \{0, 1\}, t_{i,j} \geq 0, \forall i, j, k, i', j'\\
&\text{(k) } x_{i,0,0} = 1, x_{i,n+1,0} = 1, t_{i,0} = \delta_i, \forall i \in [1, \lambda].
\end{aligned}
\tag{6.2}
$$

Constraint (b) guarantees the temporal order for the execution of two dependent modules. Constraint (c) indicates that each cloud server can and only can process one module at

one time. In another word, if two modules are scheduled to the same machine, one module will not be started until the other one is finished. Since $y_{i,j}$ and $z_{i,i',j,j'}$ are two auxiliary variables, constraints (d)-(g) show that variables $y_{i,j}$ is determined by $x_{i,j,k}$, and constraints (h)-(i) indicate the value of $z_{i,i',j,j'}$ depends on the value of $t_{i,j}$ and $t_{i',j'}$. According to our system model, the input data of the application is from mobile device, so we have $x_{i,0,0} = 1$ for all i in constraint (k). Note that $t_{i,0}$ represents the *release time* of request i. Thus, we have $t_{i,0} = \delta_i$ in constraint (k).

6.2.4 Uniqueness of MCPP

We compare the MCPP with classical job scheduling problems and discuss their differences.

Comparison with TSPHC. The first classical scheduling problem similar to MCPP is Tasks Scheduling Problem for Heterogeneous Computing(TSPHC) [THW02]. In this problem, an application is represented by a Directed Acyclic Graph (DAG) in which nodes represent application tasks and edges represent intertask data dependencies. Given a heterogeneous machine environment, where the machines have different processing speeds, and the data transfer rate between machines are different, the objective of the problem is to map tasks onto the machines and order their executions so that task-precedence requirements are satisfied and a minimum completion time is obtained. The TSPHC is NP-complete in general case, and various efficient heuristics were proposed in the literatures [THW02] [LZ08].

Intuitively, we may model our problem as similar to MCPP as possible. In our problem, we have $\lambda \times n$ tasks, where the precedence dependence exists among the tasks from the same users. The machines can be abstracted as a set of r cloud servers/VMs and one mobile device. Note that the sole mobile device in our model, unlike the machines in TSPHC, is able to execute more than one task simultaneously. The data transfer rate is

infinite between the cloud VMs , while being constrained between any pair of the mobile device and cloud VM. The problem is to map the tasks onto the $(r + 1)$ machines such that the precedence constraints are satisfied, and the weighted summation of all the tasks' completion time is minimized. The tasks that appear in the last position of the application flow are assigned the weight of one, and others are assigned the weight of zero.

The key difference between MCPP and TSPHC is the optimization objective. In TSPHC the optimizion objective is the makespan which is the maximum completion time of all the tasks, while in MCPP the objective is the total weighted completion time. Although various efficient heuristics were proposed for TSPHC to optimize the makespan, there were few solutions on optimizing the total weighted completion time. We can only find some early efforts to minimize the total weighted completion time on single machine or on parallel machine without considering the communications. Even these simplified versions for MCPP have been proved to be NP-hard [HSW96].

Comparison with HFS. The second classical scheduling problem is Hybrid Flow Shop (HFS) scheduling [BB02]. In this problem, the job is divided into a series of stages. There are a number of identical machines in parallel at each stage. Each job has to be processed first at Stage 1, then Stage 2, and so on. At each stage, the job requires processing on only one machine and any machine can do. Assuming all the jobs are released at the beginning, the problem is to find a schedule to minimize the makespan. We note that the application and its functional modules in our problem are analogous to a job and stages in HFS. The mobile devices and cloud VMs may be modeled as the machines in HFS. However, our MCPP is far different from HFS in terms of the following aspects. 1) In MCPP, there exists communication overhead between stages, which makes the problem more complex than HFS; 2) in MCPP, since both cloud VM and mobile device are able to execute any module of the application, the set of machines are not partitioned into subsets according to

the stages; 3) the objective in MCPP is the total completion time rather than the makespan.

6.3 SearchAdjust

The most widely used method for solving MILP problems is branch and bound [LW02]. It transfers the MILP problem into standard Linear Programming (LP) problem by relaxing the integral variables. Based on the optimal solution obtained using LP, the MILP problem is then divided into subproblems by restricting the range of the integral variables. The subproblem is solved using LP, and then divided into sub-subproblems. This process is done recursively until a feasible and satisfactory solution is found.

Unfortunately the LP-based solution is not practical for the MCPP, because it contains an exploding number of variables and constraints when the problem scales up. From equation (2), we can see that the number of variables $z_{i,i',j,j'}$ achieves the magnitude of $\lambda^2 n^2$, and the number of constraints (c) is $\lambda^2 n^2 r$. Although we can express the model by deleting all the auxiliary variables $z_{i,i',j,j'}$ and $y_{i,j}$, the number of variables $x_{i,j,k}$ remains a large magnitude of $\lambda n r$. Thus, in this section, we design a greedy heuristic algorithm, named as *SearchAdjust*, to solve the MCPP.

6.3.1 Overview of SearchAdjust

The idea of SearchAdjust is that we first relax the resources constraints in the MCPP. For each user, we can have an optimal partitioning by using the solution of SCPP. Under these optimal partitions, we *search* the time intervals during which the resources constraints are violated. We then *adjust* the schedule in a greedy way that can release as long resources occupation period as possible at these time intervals, and meanwhile increase the average application delay as little as possible. The *searching* and *adjusting* are done alternatively until the resource constraints are satisfied for all the time. The algorithm is designed due to the observation that the SCPP initiated solution is optimal but not feasible in the solution

space of MCPP. Hence, we can adjust the initial solution iteratively to make it feasible, while not sacrificing the objective function (average application delay) too much.

Before describing the algorithm, we first introduce three important data structures as below.

• *Execution Schedule* S: For each user i, we can create a $n \times 3$ table to store its *execution schedule* in which each row is a three-tuple $(x_{i,j,0}, \tau_{i,j}, t_{i,j})$, where $x_{i,j,0}$ indicates at which side the module (i, j) is scheduled, and $\tau_{i,j}$, $t_{i,j}$ are respectively the start time and completion time of the module (i, j).

• *Cloud Resource Occupation List* L_{cro}: The list records the number of occupied servers at each time interval. Each element e of the list is denoted as $(start, end, num)$, where $start/end$ represents the start/end point of the time interval, and num is the number of occupied server at the interval. The time intervals in the list have no overlapping with each other, and are able to constitute a continuous time interval. The elements are stored in the list according to the ascending order of the time intervals. Thus, we have $e_k.end = e_{k+1}.start$ and $e_k.start < e_{k+1}.start$, $\forall k \geq 1$. Note that the length of each interval is not necessary to be the same.

• *Module Adjustment List* L_{adj}: The list records the modules which could release the cloud resource occupation period by waiting to execute later on or changing to run at mobile side, their rewards, and corresponding released cloud resource occupation period for the adjustment. We denote each item as $(i, j, reward, D_{rel})$, where i, j indicates the module, $reward$ represents the reward of the adjustment on this module and D_{rel} is the released cloud resource occupation period. The modules are stored in the list by a descending order of $reward$.

Algorithm 4 gives the pseudo code of the greedy heuristic. First, we get the optimal partitioning and corresponding execution schedule for each user without considering the

cloud resource constraint (line 1). Second, we compute the cloud resource occupation list L_{cro} (line 2). It records the number of the occupied/in-use servers at each time interval. Then, we find the earliest interval that the number of occupied servers exceeds the up-bound r (line 3). We name the start of the interval as **critical point** t_{cri} on the time axis, as before it the resource constraint is satisfied, and after it the constraint is violated. Next, for each user we look for the module that was scheduled at the cloud side, and the execution time of which spans the critical point. In order to release the cloud resource immediately after the critical point, we adjust the schedule of this module by moving it back to mobile side or delaying its execution for some time. The adjustments are scored based on a reward function (which represents the greedy strategy in our algorithm) (line 6). The modules with positive score/reward are added into the module adjustment list L_{adj} (line 7-9). After the searching for all the users, α modules with largest rewards are selected from the list L_{adj} to adjust (line 12). In each iteration, the critical point t_{cri} would be moved forward along the time axis. The algorithm stops until the resource constraint is satisfied at all the times (line 3).

6.3.2 Details of SearchAdjust

In the following, we present details of *SearchAdjust*: 1) how to obtain the initial optimal but infeasible solution; 2) how to compute the cloud resource occupation list L_{cro} and search the critical point; 3) the reward function of *SearchAdjust*; 4) how to determine the number of modules α to adjust in each iteration.

Initial Solution

Consider the SCPP shown in Definition 1, suppose that the completion time of module j is denoted as t_j. As every module can be completed either at mobile side or at the cloud side, we use notation $t_j^{(c)}$ to represent the completion time of module j if it is scheduled

Algorithm 4: The Greedy Heuristic for MCPP

```
Input  : A set of λ users, and a set of r cloud servers
Output: The execution schedule (x_{i,j,0}, τ_{i,j}, t_{i,j})
Compute the initial execution schedule using SCPP solution;
Compute the cloud resource occupation list L_cro ;
while search the critical point from L_cro do
    for each user do
        if find the module that is scheduled onto cloud, and its execution time cover
        the critical point then
            Compute the reward of adjusting the module;
            if Reward > 0 then
                Insert this module into list L_adj by a descending order of its reward;
            end
        end
    end
    Select the first α modules from list L_adj to adjust;
    Update the execution schedule of the selected modules;
    Re-compute the cloud resource occupation list L_cro;
end
return the execution schedule for all the users;
```

at the cloud side. Correspondingly, notation $t_j^{(m)}$ is the completion time of module j if scheduled at mobile side. Then, we have a recursive formulation of t_j:

$$t_j^{(c)} = min\{t_{j-1}^{(c)} + c_j,\ t_{j-1}^{(m)} + w_j + \pi_{j-1,j}\}, \tag{6.3}$$

$$t_j^{(m)} = min\{t_{j-1}^{(m)} + w_j,\ t_{j-1}^{(c)} + c_j + \pi_{j-1,j}\}, \tag{6.4}$$

where $j = 1, 2, ..., n, n+1$. The module 0 and module $n+1$ are respectively the entry and exit module we have added virtually into the application graph. The computation time of these two modules are zero.

In order to determine the optimal partitioning, we construct a graph which contains $2 \times (n+1)$ nodes. Each node is denoted as $v_j^{(p)}$, and labeled with its completion time $t_j^{(p)}$, where $0 \leq j \leq n+1$ and $p \in \{c, m\}$. Since the input data of the application is from the mobile device, we let $t_0^{(m)} = 0$ and $t_0^{(c)} = \infty$. Starting from the nodes $v_0^{(m)}$ and $v_0^{(c)}$, and we can recursively compute the labels of all the nodes by Equation (6.3)(6.4). For each node,

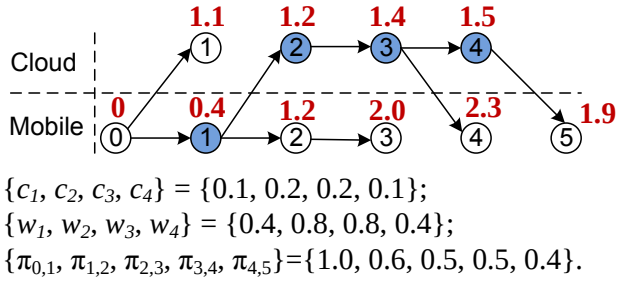


Fig. 6.3: An example of SCPP solution

for example, $v_j^{(m)}$, there are two possible edges from its precedent nodes to it, $v_{j-1}^{(c)}$ and $v_{j-1}^{(m)}$. The edge that leads to less value of $t_j^{(m)}$ according to Equation (6.4) is added into the graph. The partitioning result is actually a path from node $v_0^{(m)}$ to node $v_{n+1}^{(m)}$. Fig.6.3 shows an example of the method. There are four modules in the application. The colored nodes indicate the places that the modules are scheduled to. For the MCPP, with the optimal partitions $x_{i,j,0}$ for each user i, we can easily obtain the initial *execution schedule* $S = \{(x_{i,j,0}, \tau_{i,j}, t_{i,j})\}$.

Computation of Cloud Resources Occupation List and Critical Point

The cloud resource occupation list L_{cro} records the number of occupied server at each time interval. In each iteration of Algorithm 4, L_{cro} needs to be re-computed. We design an algorithm to calculate L_{cro} which is shown in Algorithm 5. The input of the algorithm is the *execution schedule* of each user $\{(x_{i,j,0}, \tau_{i,j}, t_{i,j}) | 1 \leq j \leq n, 1 \leq i \leq \lambda\}$. In Algorithm 2, L_{cro} is first initialized by the time interval $(0, \infty)$, with the number of occupied server at this interval being zero. For each module (i, j) that is allocated onto cloud side (line 4), we first respectively search the intervals from L_{cro} in which the start and completion time of the module's execution period are located (line 5-6). The interval covering the start or the completion point is split into new sub-interval, which are then inserted into L_{cro} (line 12,15,21). For the interval which is entirely covered by modules (i, j)'s execution period,

we increase the number of its occupied server by one (line 9). The algorithm stop until all the modules on cloud are finished. After finishing one module, the length of L_{cro} increases at most by 2. The length of L_{cro} returned by Algorithm 5 would be at most $2\lambda n$. The time complexity of the algorithm is on the order of $O(\lambda)$.

Algorithm 5: The Algorithm for Computation of Cloud Resource Occupation List

Input : $\{(x_{i,j,0}, \tau_{i,j}, t_{i,j}) | 1 \leq j \leq n, 1 \leq i \leq \lambda\}$
Output: L_{cro}
1 $L_{cro} \leftarrow \{(0, \infty, 0)\}$;
2 **for** *each user* i **do**
3 **for** *each module* j **do**
4 **if** $x_{i,j,0} == 1$ **then**
5 Search the element e_s in L_{cro} such that $e_s.start < \tau_{i,j} \leq e_s.end$;
6 Search the element e_v such that $e_v.start \leq t_{i,j} < e_v.end$;
7 **if** $s < v$ **then**
8 **for** k-*th element in* L_{cro}, $s < k < v$ **do**
9 $e_k.num \leftarrow e_k.num + 1$;
10 **end**
11 **if** $\tau_{i,j} \neq e_s.end$ **then**
12 Insert element $(\tau_{i,j}, e_s.end, e_s.num + 1)$ into L_{cro} after e_s;
13 **end**
14 **if** $t_{i,j} \neq e_v.start$ **then**
15 Insert element $(e_v.start, t_{i,j}, e_v.num + 1)$ into L_{cro} before e_v;
16 **end**
17 $e_s.end \leftarrow \tau_{i,j}$;
18 $e_v.start \leftarrow t_{i,j}$;
19 **end**
20 **else**
21 Insert elements $(\tau_{i,j}, t_{i,j}, e_s.num + 1)$ and $(t_{i,j}, e_s.end, e_s.num)$ into L_{cro} after e_s;
22 $e_s.end \leftarrow \tau_{i,j}$;
23 **end**
24 **end**
25 **end**
26 **end**
27 **return** L_{cro};

Critical point t_{cri} is defined as the start time of the first time interval in L_{cro} during which the number of occupied servers exceed the up-bound r. Based on the L_{cro}, it is easy to find the *critical point* t_{cri}. The whole time axis are divided into two periods by t_{cri}. In the period before the time t_{cri}, the number of occupied servers is lower then the up-bound

r, while during the period after L_{cro} the number of occupied servers exceed the up-bound r. In Algorithm 4, t_{cri} is put forward on the time axis in each iteration until no t_{cri} is found from L_{cro}.

Reward Functions/Greedy Strategies

The reward function is to evaluate the reward of adjusting the schedule of one module. It is defined by the *released cloud resource occupation period*, denoted as D_{rel}, minus the *extra delay* caused by this adjustment, denoted as D_{delay},

$$\textbf{Reward} = D_{rel} - D_{delay}. \tag{6.5}$$

The reward function is defined due to the motivation that we always prefer to select the module to adjust which can release as a long cloud resources occupation period as possible, and meanwhile causing as short extra delay as possible. Next we describe the definition of D_{rel} and D_{delay}.

Released Cloud Resource Occupation Period. Note that only the modules satisfying the following two conditions could be adjusted: (a) the module is executed at the cloud side, $x_{i,j,0} = 0$; and (b) its execution duration covers the critical point, $\tau_{i,j} < t_{cri} < t_{i,j}$. For one user i, assuming the module j_0 is the candidate module which cloud be moved to mobile side. To distinguish with the original *execution schedule* of user i, we use $\tau'_{i,j}$ and $t'_{i,j}$ to respectively represent the start time and completion time after the movement of module j_0. The *released cloud resource occupation period* due to the adjustment of module (i, j_0) is defined by

$$D_{rel}(i, j_0) = min\{\tau'_{i,j_c}, t_{i,j_m-1}\} - t_{cri} \tag{6.6}$$

where $j_c, j_m \in [j_0 + 1, n + 1]$. j_c represents the first successor of module j_0 that is scheduled

to the cloud; if no successor of module j is at the cloud side, then $\tau'_{i,j_c} = \infty$. j_m is the first successor of module j_0 that is scheduled onto the mobile side; if no successor of module j_0 is at mobile side, then $j_m = n + 1$.

Extra Delay. The extra delay caused by the adjustment of module (i, j_0) is defined by

$$D_{delay}(i, j_0) = \tau'_{i,j_0+1} - \tau_{i,j_0+1} \tag{6.7}$$

The reward of adjustment of one module could be positive or negative. Fig.6.4(a) indicates that $D_{rel} > D_{delay}$, hence the reward of the adjustment is positive; while in Fig.6.4(b) the reward of the adjustment is negative. In each iteration of algorithm, we only select the modules with positive and as large as possible reward to adjust.

Remember that we actually have two adjusting options to release the resources which have been occupied at the critical point, i.e., **waiting** and **movement**. Waiting means to purely delay the execution of the modules at the cloud side, while movement means to change the execution place of the module. However, as shown in Fig.6.4(c), we can see that waiting adjustment always has a non-positive reward. Hence, in our algorithm we do not consider the waiting adjustment.

Other Reward Functions. Now we pose another question: do we have other reward functions? Actually there exist two typical functions: (a) **Reward** $= -D_{delay}$, and (b) **Reward** $= D_{rel}$. The former function means that the modules with as small extra delay as possible are selected despite of its released cloud resource occupation duration. The latter function prefers to select the module that could release longer cloud resource occupation duration. We evaluate the two functions in Section 6.6. We find that function (a) obtains good average application delay, but requires a long time to converge, while function (b) leads to bad average application delay. The reward function in Equation 6.5 is able to achieve good average delay and fast convergence speed.

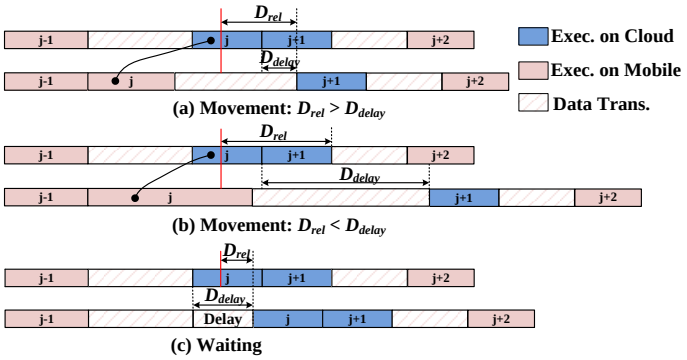


Fig. 6.4: Reward function

Number of modules to adjust α

Now we answer the question: how many modules are adjusted in each iteration? Note that in Algorithm 4, only the modules with positive rewards are put into the L_{adj}. First, we get average D_{rel} of all the modules in L_{adj}, which is denoted as $D_{rel}^{(avg)}$. Then, from the cloud resource occupation list L_{cro}, we compute the average number of occupied servers in the period from t_{cri} to $t_{cri} + D_{rel}^{(avg)}$, which is denoted as $num^{(avg)}$. The number of modules to adjust α is given by Equation (6.8-6.10):

$$\alpha = \min\{\text{LengthOf}(L_{adj}), num^{(avg)} - r\}, \tag{6.8}$$

$$num^{(avg)} = \frac{\sum_{e \in L_{cro}^{(sub)}} (e.end - e.start) \times e.num}{\sum_{e \in Sub\{L_{cro}\}} (e.end - e.start)}, \tag{6.9}$$

where $L_{cro}^{(sub)}$ is the subset of L_{cro}, which includes all the time intervals located at $[t_{cri}, t_{cri} + D_{rel}^{(avg)}]$,

$$L_{cro}^{(sub)} = \{e \in L_{cro} | e.start, e.end \in [t_{cri}, t_{cri} + D_{rel}^{(avg)}]\}. \tag{6.10}$$

In our algorithm, the adjustment on the schedule usually leads to the decreasing of application performance. We avoid the case that excessive modules are moved back to

mobile side, so that the cloud servers are not utilized completely. So α is constrained by an up-bound $num^{(avg)} - r$ as shown in Equation (6.8).

6.3.3 Theoretical Analysis

In Algorithm 4, the execution schedules indicate if each module is executed at mobile side or at the cloud side. However, for the modules that are allocated to the cloud, the results do not specify which cloud server hosts the offloaded module. We may question: could the completion time of each module $t_{i,j}$ be delayed when allocating the offloaded modules onto the cloud servers?

Theorem 1 (Feasibility) For the execution schedule $S = \{(x_{i,j,0}, \tau_{i,j}, t_{i,j})\}$ generated by Algorithm 4, we can always find a feasible schedule $S' = \{(x'_{i,j,k}, \tau'_{i,j}, t'_{i,j})\}$ of MCPP by assigning the offloaded modules onto the cloud servers, such that each module (i, j) is completed no later than $t_{i,j}$, $t'_{i,j} \leq t_{i,j}$.

Proof. Consider a simple case where the offloaded modules are scheduled online to the cloud servers with the policy of 'first-come-first-serve'. $\tau_{i,j}$ can be taken as the time that the module (i, j) comes to the cloud. We can prove using mathematical induction, by the 'first-come-first-serve' policy every offloaded module (i, j) can start exactly at time $\tau_{i,j}$ and be completed by the time $t_{i,j}$ on the cloud servers. For the new schedule S', we have $\tau'_{i,j} = \tau_{i,j}$, and $t'_{i,j} = t_{i,j}$.

The offloaded modules are ordered according to their arriving time $\tau_{i,j}$. For the 1-st module which comes to the cloud earliest, obviously it is able to start at its arriving time τ_{i_1,j_1}. For the s-th module which comes to the cloud in s-earliest time, we can prove if 1-st, 2-nd, ... , $(s-1)$-th modules start to run at their arriving time, then the s-th module can also be executed at the cloud servers at its arriving time τ_{i_s,j_s}. The proof is as follows.

Up to the time when the s-th module comes, we can conclude: 1) at least one cloud server is idle at $\tau_{i,j}$; 2) if the cloud server is idle at time τ_{i_s,j_s}, it must be idle all the time $[\tau_{i_s,j_s}, +\infty)$. The first conclusion is due to the fact, that our algorithm guarantees the number of occupied cloud servers does not exceed the constraint r at all the times if each module is executed according to the schedule S. The second conclusion is proved in this way: if 2) does not hold, which means that up to the time τ_{i_s,j_s}, some module which arrives at the cloud earlier than the s-th module has already been allocated to the cloud server. It contradicts with the 'first-come-first-serve' policy. Because of the two conclusions, the s-th module can start on the cloud server at its arriving time. If multiple idle servers exist, we randomly allocate the module to one of them. □

Theorem 2 (Complexity) For a given application graph, the complexity of Algorithm 4 is $O(\lambda^2)$.

Proof. In Algorithm 4, the evaluation of the reward for adjusting each offloaded module is the most time costly operation. In each iteration, at most $\lambda \times n$ modules need to be evaluated. The question is how many iterations Algorithm 4 needs to stop. The worst case is that the resources constraints $r = 0$, and only one module is moved to the mobile side in each iteration. In this case, Algorithm 4 needs at most $\lambda \times n$ iterations, such that the offloaded modules are all moved to the mobile side. By neglecting the constant n for a given application graph, the worst time complexity of Algorithm 4 is on the order of $O(\lambda^2)$. □

6.4 Benchmark Offline Solutions

6.4.1 List Scheduling (LS) based Solutions

List scheduling is considered as an efficient method to solve existing job scheduling problems [BB02] [LZ11] [AK96] [DA98]. Although MCPP is different from existing job scheduling problems, we are still interested to know how list scheduling based algorithms perform when used to solve the MCPP. We have two ways to solve the MCPP using list scheduling method. One way is, as described in 3.4, we can fit the MCPP into the TSPHC model by abstracting both the mobile devices and cloud servers as the processors. The problem can be solved by Heterogeneous-Earliest-Finish-Time (**HEFT**) algorithm, which is demonstrated to be an accurate and efficient list scheduling algorithm for TSPHC [THW02]. The time complexity of HEFT is on the order of $O(\lambda \times r)$. We will evaluate the performance of HEFT algorithm when it is used to solve the MCPP in Section 6.6.

The other way is that we divide the MCPP into two phases. In the first phase, named as partitioning, we simply decide for each user which modules are executed at mobile side and which others are scheduled at the cloud side. The partitioning phase is done using the SCPP method, which is introduced in Section 6.3. In the second phase, the list scheduling method is applied to allocate the offloaded modules onto the cloud servers or to move the offloaded modules to the mobile side. Based on the generated execution schedule from the

Algorithm 6: The MEDLS Heuristic

Input : The execution schedule $(x_{i,j,0}, \tau_{i,j}, t_{i,j})$
Output: The execution schedule $x_{i,j,k}, \tau_{i,j}, t_{i,j}$

```
for each user i do
    Insert the first offloaded module of the user into a scheduling list L_s;
end
Sort the modules in L_s by non-increasing order of their ready time τ_{i,j};
while there are unscheduled tasks in the list L_s do
    Select the first module (i_0, j_0) from the list for scheduling;
    for each machine k, including cloud servers and the user's mobile device do
        Compute the extra delay of module (i_0, j_0) on machine k. // If k = 0, the
        extra delay is obtained by Equation(7), otherwise the extra delay is
        calculated by the earliest time that the module can actually start on the
        cloud server minus the module's ready time τ_{i,j};
    end
    Assign task (i_0, j_0) to the machine that minimizes the extra delay;
    Update the execution schedule for user i_0, (x_{i_0,j,k}, τ_{i_0,j}, t_{i_0,j});
    Remove the module (i_0, j_0) from L_s, and add the first successive offloaded
    module of (i_0, j_0) into L_s;
end
return the execution schedule x_{i,j,k}, τ_{i,j}, t_{i,j};
```

first phase, the task that is ready to start earliest is assigned with the highest priority. Each selected task will be scheduled to the machine (including the cloud servers and the mobile device) which leads to a minimum extra delay. We name the method as Minimum Extra Delay List Scheduling (**MEDLS**). Algorithm 6 gives the pseudo code of the MEDLS. The time complexity of the algorithm is also $O(\lambda \times r)$.

6.4.2 Hybrid Method of SearchAdjust and List Scheduling

In the greedy heuristic algorithm, *SearchAdjust*, by an iterative adjustment on the SCPP initialized execution schedule, the number of occupied resources et each time does not exceed the resources constraint r. In MEDLS algorithm, the SCPP initialized execution schedule is also adjusted by the list scheduling method. Both algorithms generate feasible schedules by doing the adjustments on the SCPP initialized solution. However, greedy heuristics is by doing the user-oriented adjustment, while MEDLS is through the machine-oriented adjustment. Intuitively, the two types of adjustment can be combined in the way: first, the

user-oriented adjustment by Algorithm 4 is performed by relaxing the resource constraint r to $(1+\gamma)r$; then, the machine-oriented adjustment by Algorithm 6 is performed. We call the combined method as γ-**Greedy** algorithm. We will evaluate the performance of this algorithm under different parameter of γ.

6.5 Online Solution

In contrast to the offline solutions, an online solution only knows the release time of past requests and current requests but have no knowledge about the future requests. The partitioning for one user's request can not be determined before the request is released. In this section, we present the design, and analyze the performance of our online solution.

In our online solution, we do not partition user's requests one by one. Instead, we divide the whole time interval $(0, T)$ into small time slots, and do the partitioning every time slot. Let η denotes the index of the time slot, and Δt denotes the length of the time slot. We do the partitioning at the end of each time slot for all the requests that are released during that time slot. For each partitioning, we first select a number of idle servers from all the r cloud servers, and then use our offline solution to do the partitioning with the selected cloud servers. The offline solution is performed repeatedly every Δt. Note that Δt is small enough relative to the completion time of the application.

Now the question here is that how many servers are allocated to the load at each time slot η, such that the overall delay for the requests during $(0, T)$ is as low as possible. If we allocate too many servers to current load, it is possible that there is no enough idle cloud servers to accommodate the load in future time slots. If we always try to reserve more servers to future load, the performance of current load would yield to significant degradation. The online algorithm tries to balance a tradeoff between provisioning enough servers for current load, and reserving enough servers for future load, through a control parameter Λ.

Let λ_η denote the number of requests that arrive at the system during time slot η. Let r_η denote the number of servers that are allocated to the requests at the end of time slot η. The overall delay of the λ_η requests are obtained by our PRL model $d_\eta = \mathcal{F}(\frac{r_\eta}{\lambda_\eta})$, which is numerically analyzed in Section 6.6. Suppose d_η is normalized by the length of the time slot Δt. Think in the way that in order to accommodate the load λ_η, r_η servers will be occupied for d_η time slots. Therefore, we define *workload size* W_η that arrives at the cloud servers at the end of time slot η by $W_\eta = r_\eta \times d_\eta = r_\eta \times \mathcal{F}(\frac{r_\eta}{\lambda_\eta})$. Note that the workload size W_η reduces by r_η at the end of time slot $\eta + 1$. Let Q_η denote the total backlogged workload size of the cloud servers at the end of time slot η, before any other loads arrive. Let D_η denote the number of servers that are busy/occupied at the end of time slot η, where $0 \leq D_\eta \leq r$. Then the dynamic of $Q_{\eta+1}$ can be described as

$$Q_{\eta+1} = Q_\eta + W_\eta - D_\eta. \tag{6.11}$$

Without loss of generality, we assume load λ_η is a stochastic process across time slot η. Let $\overline{\lambda}$ and var respectively denote the expectation value and variance of load sequence λ_η. We say that the system is stable if $\lim_{\eta\to\infty} E(Q_\eta) < \infty$, i.e., the amount of backlogged workload size is bounded. The arriving load λ_η is said to be supportable if there exists a resource allocation mechanism under which the system is stable.

Theorem 3 For any given $\overline{\lambda}$ and r, the expectation of application delay of the arriving load λ_η is up-bounded by d_{opt}, where

$$\begin{aligned} d_{opt} &= \min_{r^* \geq 0} \mathcal{F}(\frac{r^*}{\overline{\lambda}}), \\ \text{s.t. } & r^* \times \mathcal{F}(\frac{r^*}{\overline{\lambda}}) < r. \end{aligned} \tag{6.12}$$

Proof. In order to support the arriving load, it should be satisfied that $\lim_{\eta\to\infty} E(Q_\eta) < \infty$. Thus, from Equation (6.11), we have $E(W_\eta) = E[r_\eta \times \mathcal{F}(\frac{r_\eta}{r_\eta})] \leq D_\eta$. D_η represents the *workload size* that the cloud can finish in time slot η. The maximum of D_η is equal to r,

and can be achieved if and only if all the r cloud servers are busy to process the workload during time slot η. Thus, we have $E[r_\eta \times \mathcal{F}(\frac{r_\eta}{r_\eta})] \leq r$. Let $E(r_\eta) = r^*$, we then get Equation (6.12). □

The key aspect of our algorithm is that it manages a pool of idle servers. At each time slot, some servers are removed from the pool to accommodate the coming load, meanwhile some new servers become idle and are added into the pool. Suppose I_η is the number of idle servers at the end of time slot η before allocating the servers for load λ_η. The parameter I_η can be obtained by the recursive equation

$$I_{\eta+1} = I_\eta - r_\eta + R_{\eta+1}, \tag{6.13}$$

where r_η are the number of servers that are allocated to load λ_η, and $R_{\eta+1}$ are the number of servers which are newly added into the pool of idle servers at the end of time slot $\eta + 1$. Note that we have $I_1 = r$ at the initial time slot.

The **online algorithm** first calculates the optimal overall delay d_{opt} and corresponding number of allocated servers r_{opt} at each time slot according to Equation (6.12), where $r_{opt} = argmin_{r^* \geq 0} \mathcal{F}(\frac{r^*}{\overline{\lambda}})$. At the end of each time slot η, the algorithm does the following:

- Compute the number of idle servers I_η by Equation (6.13).
- Compute the number of released servers in next time slot $R_{\eta+1}$ according to the delay of previous load $d_{\eta-1}, d_{\eta-2}, d_{\eta-3}$ ...
- Determine the number of servers to be allocated r_η. We are trying to guarantee that the delay of the load λ_η achieves d_{opt}. Thus, intuitively we would allocate $\frac{r_{opt}}{\overline{\lambda}} \times \lambda_\eta$ servers to the current load. If the number of idle servers is not enough to guarantee the delay d_{opt}, i.e., $I_\eta < \frac{r_{opt}}{\overline{\lambda}} \times \lambda_\eta$, we would allocate all the idle servers to current load, i.e., $r_\eta = I_\eta$; otherwise, we would consider the following rules:

- **Rule 1** Allocating more servers to improve the performance at current time slot, i.e., $r_\eta \geq \frac{r_{opt}}{\overline{\lambda}} \times \lambda_\eta$.

- **Rule 2** Reserving enough servers for next time slot. The idle servers in next time slot $\eta+1$ would be enough to accommodate $(1+\Lambda)\overline{\lambda}$ load, i.e., $r_\eta < I_\eta + R_{\eta+1} - (1+\Lambda)\overline{\lambda}$.

- **Rule 3** Avoiding over-provisioning servers for current load, i.e., $r_\eta \leq C_{max}\lambda_\eta$, where C_{max} is a constant of our PRL model. If and only if $\frac{r_\eta}{\lambda_\eta} < C_{max}$, increasing the number of allocated servers r_η would lower the delay of current load (see Fig.6.5f); otherwise if $\frac{r_\eta}{\lambda_\eta} \geq C_{max}$, increasing r_η would not improve the performance of current load.

• If Rule 1 contradicts with Rule 2, i.e., $\frac{r_{opt}}{\overline{\lambda}} \times \lambda_\eta > I_\eta + R_{\eta+1} - (1+\Lambda)\overline{\lambda}$, then we give priority to Rule 1, in which case we have $r_\eta = \frac{r_{opt}}{\overline{\lambda}} \times \lambda_\eta$; otherwise we have $r_\eta = \min\{I_\eta + R_{\eta+1} - (1+\Lambda)\overline{\lambda}, C_{max}\lambda_\eta, I_\eta\}$. Note that Rule 1 and Rule 3 never contradicts with each other, because $\frac{r_{opt}}{\overline{\lambda}} \leq C_{max}$.

6.6 Evaluation

We will respectively evaluate the performance of the offline solutions and online solutions. Our online solution divides the time axis into a number of small time slots, in each of which the offline solution is applied to partition all the requests that arrive during that time slot. Since the time slot is small enough, the release time of all the requests in each time slot is the same. Therefore, for simplicity, in the evaluation of offline solution, the release time of all the requests are assigned as zero. Through the evaluation of various offline solutions, we aim to answer two questions: 1) which solution performs best; 2) how the application performance (delay) varies depending on the number of cloud resources and the load, i.e., Performance-Resource-Load (PRL) Model. The evaluation of online solution is then based on the PRL model and the real world wikipedia load traces [UPS09].

6.6.1 Evaluation of Offline Solutions

We use the application of image based object recognition as shown in Fig.6.1 in our evaluation. It contains seven modules, therefore we have $n = 7$. We profile manually the execution time of each module on our laboratory server, and the data size that needs to be transferred between two connective modules. We assume that the processing time of each module on the mobile devices is F times greater than that on the server. Since the users' mobile devices have different processing capability, the users have various factor F. In our experiments, the local computation cost is generated by $W_{\lambda\times 7} = [F_1, F_2, ..., F_\lambda]^T \times \mathcal{C}$, where $\mathcal{C}$ is a 1×7 vector of the profiled execution time of each module on the server and F_i yields a uniform distribution in the interval $[1, 6]$. The communication cost is generated by $\Pi_{\lambda\times 8} = [\frac{1}{B_1}, \frac{1}{B_2}, ..., \frac{1}{B_\lambda}]^T \times \mathcal{D}$, where $\mathcal{D}$ is a 1×8 vector of the profiled data size, and B_i $(1 \leq i \leq \lambda)$ is the communication bandwidth which also yields a uniform distribution. We have generated more than 2000 test cases by changing the number of cloud servers r or the number of users (load) λ. Whenever λ is changed, $W_{\lambda\times 7}$ and $\Pi_{\lambda\times 8}$ need to be re-generated.

The comparison of various algorithms are based on the following two metrics:

• **Metric 1: Application Delay Ratio (ADR)**. The main performance measure of the algorithms is the average application delay that is experienced by the mobile users. Since a large set of tests are performed under different load λ and resources r, it is necessary to normalize the application delay to a lower bound, which is called the Application Delay Ratio (ADR). The ADR value of an algorithm is defined by

$$ADR = \frac{application\ delay}{d_{scpp}}. \tag{6.14}$$

The denominator is the application delay under the SCPP solution. In SCPP, the cloud resources are assumed to be unconstrained, and each user's execution schedule is generated independently by SCPP method. The ADR of the MCPP algorithms can not be less than

one since the dominator is the lower bound. The MCPP algorithm that gives the lowest ADR is the best algorithm with respect to performance.

• **Metric 2: Running Time of the Algorithm**. The running time of an algorithm is its execution time for outputting the schedule. The metric gives the average cost of the algorithm. For the algorithms which have very close ADR values, the one with minimum running time is considered as the best one.

Performance of SearchAdjust

We compare the ADR performance of the greedy heuristic, SearchAdjust (Algorithm 4), under three different greedy strategies, with two list scheduling algorithms, HEFT [THW02] and MEDLS (Algorithm 6). A concise description about the algorithms is as follows.

- **G-MaxREL**. G-MaxREL is the greedy heuristic with the strategy to maximize the Released Cloud Resource Occupied Period. The reward function in the algorithm is $Reward = D_{rel}$.

- **G-MinED**. G-MinED is the greedy heuristic with the strategy to minimize the Extra Delay. The reward function is $Reward = -D_{delay}$.

- **G-MaxRME**. G-MaxRME is the greedy heuristic with the strategy to maximize Released Cloud Resource Occupied Period minus Extra delay. The reward function is $Reward = D_{rel} - D_{delay}$.

- **HEFT**. HEFT is a well-known list scheduling algorithm specifically for TSPHC problems. We can also use the algorithm to solve our MCP problem.

- **MEDLS**. MEDLS is a list scheduling algorithm which is designed to solve the MCP problem [THW02]. In this algorithm, each module is scheduled to the machine which causes the Minimum Extra Delay.

In the first experiment, the number of users is fixed at $\lambda = 2000$. The ADR-based performance of the algorithms are compared with respect to various number of cloud servers (see Fig.6.5a). Among the three greedy heuristics, G-MaxRME and G-MinED achieve better performance than G-MaxREL. Compared with HEFT, the performance of our proposed greedy heuristics (G-MaxRME and G-MinED) is better for any number of cloud servers. Compared with MEDLS, the greedy heuristics (G-MaxRME and G-MinED) have better performance when the cloud resources is relatively tight ($r < 800$). When the cloud resources increase to $r > 800$, the greedy heuristics have the same performance with MEDLS, because in this case the SCPP solution used as the initial solution in both the greedy heuristics and MEDLS becomes feasible for MCPP. The average ADR value of the greedy heuristic (G-MaxRME or G-MinED) on all the numbers of cloud servers is better than the HEFT algorithm by 11 percent, and the MEDLS algorithm by 10 percent. It is also shown that, for all these five algorithms, the performance increases as the number of the cloud servers increases. It demonstrates the application providers can increase the overall application performance by leasing more cloud resources.

Next, we fix the number of cloud servers, $r = 200$, and compare the ADR performance of the five algorithms when the number of users λ varies (see Fig.6.5b). The two proposed greedy heuristics (G-MaxRME and G-MinED) outperform other algorithms in terms of the overall ADR performance under various number of users. For the greedy heuristics and MEDLS, there exists a threshold, $\lambda = 400$, below which the performance is not affected by the number of users. It is because in this case the cloud always has enough resources to accommodate the offloaded modules, such that each user can realize its SCPP based optimal partitions. However, when the number of users exceeds the threshold, it is shown that the performance degrades quickly as λ increases.

We compare the cost of the algorithms by using the metric of the algorithm running time (see Fig.5c). MIDLS is the most costly one among the five algorithms. For the two greedy heuristics (G-MaxRME and G-MinED) which have the best ADR-based performance, it is shown that G-MaxRME is more costly than G-MinED. This is because G-MaxRME includes the released cloud resource occupation time into the reward function, and hence needs fewer iterations than G-MinED. *We conclude that G-MaxRME is the best one among all the five algorithms in terms of both ADR based performance and running time.* Another interesting thing we can observe from Fig.5c is that, our proposed greedy heuristics have less running time as the cloud resources increase, while HEFT has longer running time as the cloud resources increase. This is because the greedy heuristics need more iterations to adjust the SCPP initial solution as the cloud resources constraint is lower. However, as the cloud resources increase, the running time of HEFT increases because more time is spent in machine selection with an insertion-based policy.

Performance of γ-Greedy

In this experiment, we evaluate the effect of parameter γ to performance of the hybrid algorithm, γ-Greedy. Fig.6.6a shows the effect of parameter γ onto ADR-performance of the algorithm. It is shown that the optimal value of γ for the ADR performance of the algorithm is typically in $[0, 0.5]$, which depends on the number of provisioned cloud servers. Considering the average ADR performance with respect to various number cloud resources, the optimal γ is 0.2. It is also observed that the more the provisioning cloud resources are, the less impact the parameter γ has onto the algorithms. To explain the rationale behind this observation, it is worth noting the fact that the hybrid algorithm is the combination of our proposed *SearchAdjust* and MEDLS. There are two special cases for the value of γ. The first case is when $\gamma = 0$, the hybrid algorithm is actually the same as the *SearchAdjust*. The other case is when γ becomes large enough, the hybrid algorithm is the same as MEDLS.

It has been demonstrated in Fig.6.5a that as the cloud resources increases, the difference between the *SearchAdjust* and MEDLS is reduced in term of both the ADR performance.

We compare the hybrid algorithm γ-Greedy with *SearchAdjust*, HEFT and MEDLS. In this evaluation, the 'MaxRME' strategy is applied in both *SearchAdjust* and the hybrid algorithm. The parameter γ is assigned with the value 0.2 for the hybrid algorithm, because it is demonstrated to be optimal for the ADR performance. Fig.6.6c and Fig.6.6d are results for the comparison in terms of ADR performance and running time. The figures show that, *the hybrid algorithm has little improvement in ADR performance over SearchAdjust, but it is less costly than SearchAdjust by 27 percents.*

Performance-Resource-Load (PRL) Model

Finally, we evaluate how the ADR-based performance changes with the provisioning cloud resources r and load λ on the system by using our proposed greedy heuristic (G-MaxRME). We have measured the ADR value of G-MaxRME algorithm in about 2000 test cases, where the number of cloud servers r varies in $[10, 2000]$ and the number of users λ varies in $[200, 2000]$. Fig.6.5d and Fig.6.5e show a 3-dimension visual PRL curve and its corresponding contour lines. Note the contour line contains the (r, λ) points which has the same ADR value. The straight contour lines approximately from the origin $(0, 0)$ show that the ADR based performance depends on the ratio of r and λ. Based on this observation, we construct a mathematical PRL model $d = \mathcal{F}(\frac{r}{\lambda})$. Fig.6.5f shows the fitting curve of ADR and $\frac{r}{\lambda}$ values from our simulation results. The greater the r/λ value is, the better the performance is. When r/λ is more than about $1/2$, the performance is not affected by the increase of r/λ, because the number of cloud resources is equivalent to unlimited with respect to the number of requesting users.

Table 6.2: Parameters setting up for online algorithm

Parameters	Values
Length of time slot Δt	100 ms
Length of load trace N	3000 time slots
The expectation value of load trace $\overline{\lambda}$	305
The variance of load trace var	(5, 50)
The number of cloud servers r	3000
Control parameter of online algorithm Λ	0, 0.4

6.6.2 Evaluation of Online Solution

To realistically evaluate the performance of our online solution, we use the wikipedia request traces [UPS09] to do the simulations. The whole data set contains 10% of all the requests directed to Wikipedia server from September 19, 2007 to January 2, 2008. The total number of the requests are 20.6 billion. Each request in the data set includes a time stamp which is recorded in milliseconds. From the whole data set we select 10 traces, each of which has a length of 5 minutes. For each trace, we count the number of requests every one time slot (we set the length of time slot $\Delta t = 100ms$ in this simulation). Thus, each trace contains 3000 time slots. Note that to evaluate how the variance of the load trace influences our online algorithm, we select the 10 traces which have the same expectation value ($\overline{\lambda} = 305$) but different variances ($5 \leq var \leq 50$). Fig.6.7 shows one of the 10 selected traces. The expectation value and variance of this trace are $\overline{\lambda} = 305$, $var = 12$. Table 6.2 shows the setting up of the parameters in our simulation.

We evaluate our online algorithm in terms of the three metrics: (i) application delay, (ii) server utilization, and (iii) Service Level Agreement (SLA) violation. *Application delay* indicates the average delay of all the requests in the load trace. *Server utilization* is defined by $U = \frac{\sum_{\eta=1}^{N}(I_\eta - r_\eta)}{rN}$, where $N = \frac{T}{\Delta t}$ is the length of load trace in time slots and $I_\eta - r_\eta$ represents the number of servers that are spare during time slot η. *SLA violation* is defined

as the percentages of requests that do not meet the delay requirements. For latency sensitive applications, the application provider requires that the delay of each request should be less than a constraint. If and only if the delay of all the requests meet the delay requirement, we say the SLA are satisfied, in which case the value of SLA violation is zero. Note the difference between metric (i) and metric (iii). The two metrics are not positively correlated. If the input load trace has low application delay, it does not necessarily have low SLA violation; and vice versa.

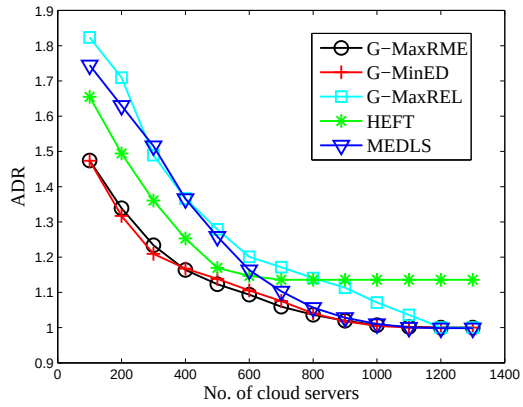
Fig.6.8 shows the evaluation results for multiple load trace variances and two values of Λ. It is shown that as the variance of input load trace *var* increases, the application delay and SLA violation generally increases and server utilization generally decreases. The reason is that as *var* increases, the load fluctuation over time has greater magnitude, resulting more time slots in which the cloud servers are either overloaded or mostly spare. In particular, Fig.6.8a demonstrates that the application delay of our online algorithm is very close to the optimum d_{opt} (shown as dash line) when the variance is small. For example, the ratio between the application delay ($var = 20$, $\Lambda = 0.4$) and d_{opt} is 1.08. The smaller *var* is, the closer the ratio value is to 1. We analyze the whole wikipedia request data set during October 2007, we found that more than 90% of the traces (3000 time slots length) has $var \leq 20$. This evaluation implies that our online algorithm can achieve a delay of $1.08d_{opt}$ at most time, i.e. 90% time of the whole month in October 2007 for wikipedia requests data set.

The trade-off between reserving more servers for future time slots ($\Lambda = 0.4$) and allocating more servers for current time slot ($\Lambda = 0$) is also interesting. Fig.6.8a shows that when the load trace has small variance, it is better to allocate more servers for current time; while the load trace has large variance, it is better to reserve more servers for future time. Although reserving more server for future time slots always leads to lower server utilization

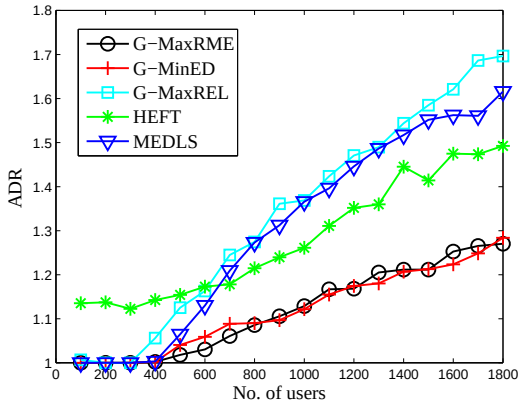
than allocating more servers for current time (shown in Fig.6.8b), it can achieve better performance in term of SLA violation (shown in Fig.6.9).

6.7 Summary

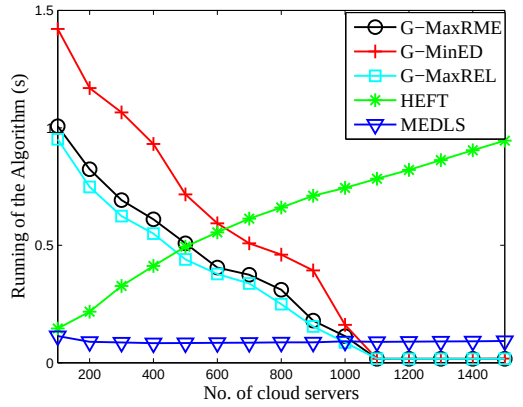
In this chapter, we have focused on the Multi-user Computation Partitioning Problem (MCPP). We have designed an offine algorithm, *SearchAdjust*, and a set of competitive benchmark algorithms to solve the problem, and conducted extensive simulations to compare their performance. *SearchAdjust* was demonstrated to outperform the list scheduling algorithms, HEFT and MEDLS, by 10 percent in term of the application delay. From the simulations, we also draw a Performance-Resource-Load (PRL) model to show how the performance (application delay) varies depending on the load and provisioned cloud resources. Based on the PRL model and offline algorithm, we further design an online algorithm that can be deployed in practical mobile cloud systems. We show our online algorithm can achieve satisfactory application delay by real trace driven simulations.



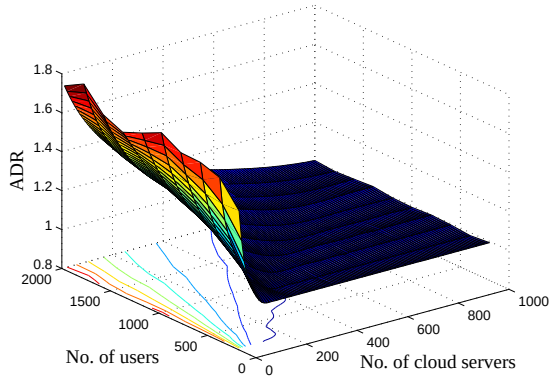
(a) Comparison of ADR based performance under various number of cloud servers ($\lambda = 2000$)



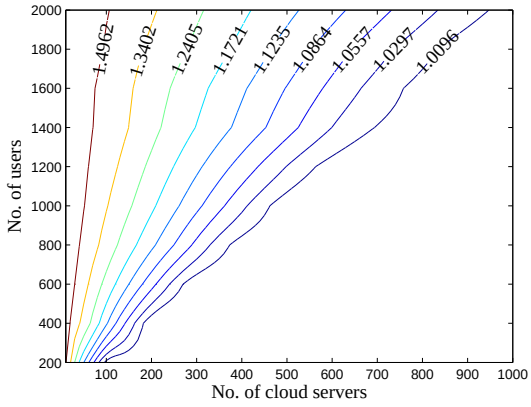
(b) Comparison of ADR based performance under various load ($r = 200$)



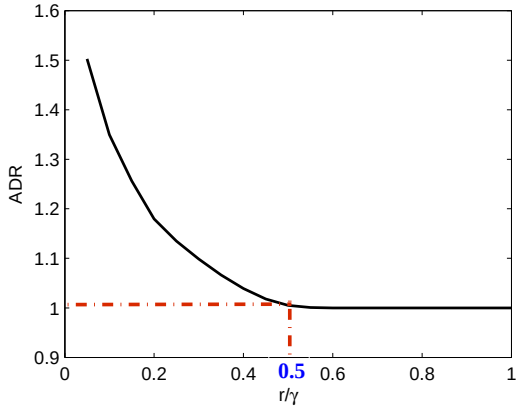
(c) Comparison of the algorithm running time



(d) The 3D Performance-Resource-Load (PRL) curve

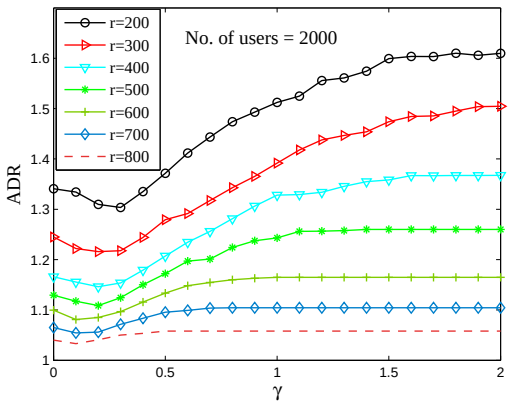


(e) Contour lines of of PRL model

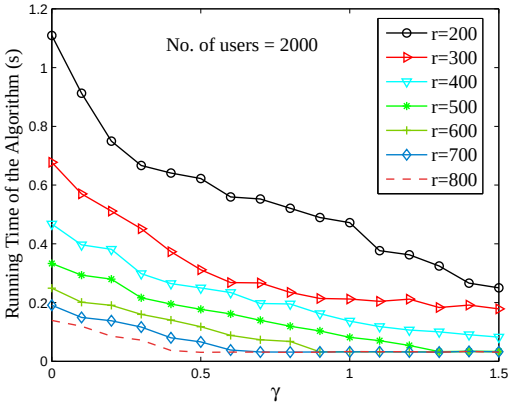


(f) The ADR performance varies depending on r/λ

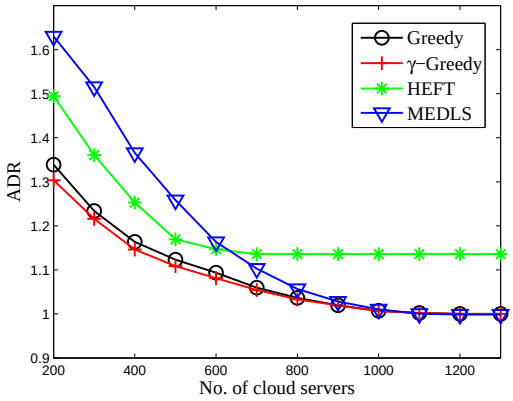
Fig. 6.5: Evaluation results of SearchAdjust



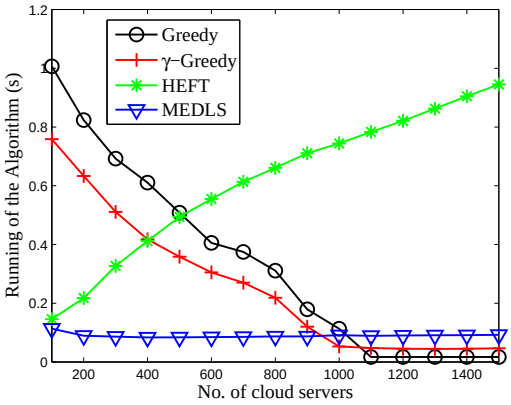
(a) The effect of γ to the ADR performance



(b) The effect of γ to the algorithm running time



(c) ADR comparison of γ-Greedy



(d) Running time comparison of γ-Greedy with other heuristics

Fig. 6.6: Evaluation results of γ-Greedy

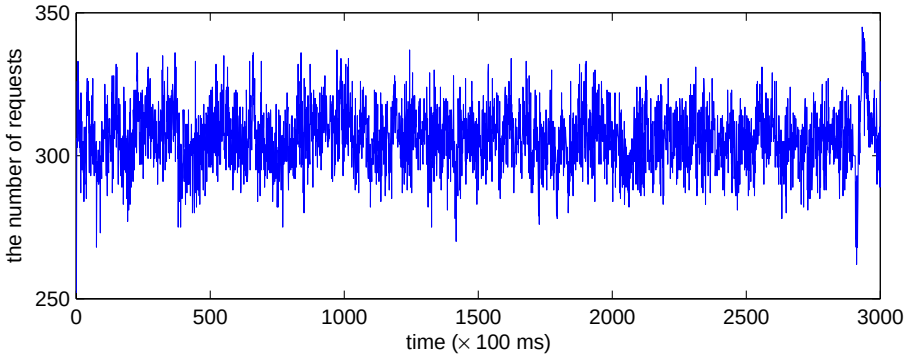
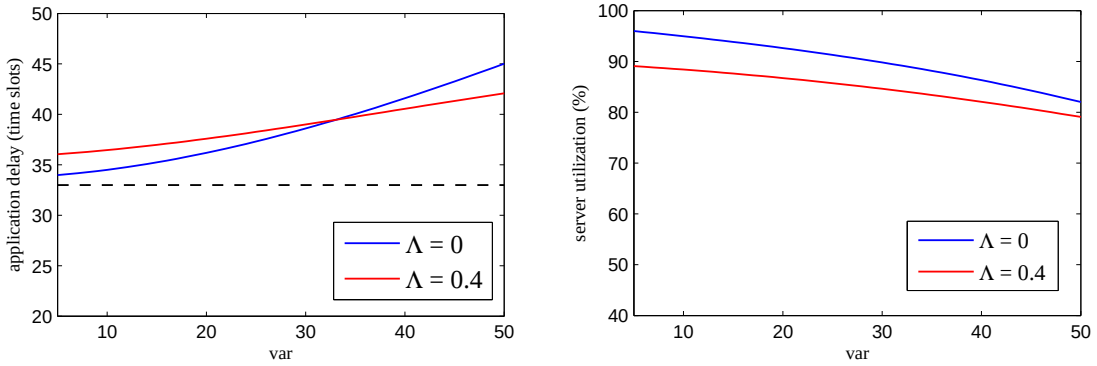


Fig. 6.7: One selected wikipedia load trace containing 3000 time slots.



(a) The overall application delay increases as load variance *var* increases

(b) The cloud server utilization varies depending on *var*

Fig. 6.8: The performance of online algorithm in term of the metrics: application delay and cloud server utilization

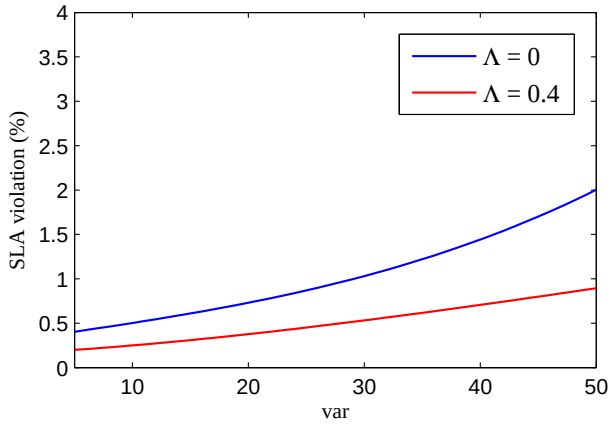


Fig. 6.9: The performance of online algorithm in term of the metrics: SLA violation

Chapter 7

Conclusions and Future Research

In this chapter, we conclude this thesis in Section 7.1 and present future research direction in Section 7.2.

7.1 Conclusions

Computation partitioning is a widely used technique for implementing high performance mobile cloud applications on today's resource constrained mobile devices. From the mobile side, it provides opportunities to augment the execution of applications on the mobile devices, by offloading intensive computations onto more powerful clouds. From the cloud provider side, it utilizes the sensing and computing capability of the mobile devices to provision new and more efficient services to the end users. Coming with great benefits of this technique, we face new challenges in ensuring high performance computation partitioning under complex properties of the applications and systems. The computation partitioning need to meet the requirement of various applications and system models.

In this thesis, we have a systematical investigation and study on computation partitioning under three models of applications and systems: computation offloading, single user computation partitioning and multiple user computation partitioning. In each model, we identified the problems which are important and not well addressed, and proposed corresponding solutions. We conclude these works as follows.

For computation offloading, we studied how to apply the principle of computation offloading into the design of accurate and efficient RFID tracking method. We proposed a hybrid method for achieving high accuracy and efficiency in object tracking based on passive RFID tags. The method can adaptively switching between using WCL and Particle Filter (PF) according to the estimated speed of the object. The method can offload complex computations involved in the particle filtering onto the nearby servers or the cloud. The offloading decisions can adjust according to the parameter of the algorithm such as the number of particles, and the network connection qualities. Through extensive simulations and two real world experiments, we showed that the hybrid RFID tracking method with computation offloading outperforms the existing methods WCL and PF, and the hybrid method without computation offloading.

For single user computation partitioning, we studied how to tackle the complexity respectively in the dimension of application and system. We focused on two problems. One problem was partitioning for data streaming applications and aimed to maximize the data processing speed (or throughput). The other problem was partitioning applications under dynamic mobile cloud environment, in which the device and network status can fluctuate during the life cycle of the application.

With respect to the first problem, we designed a framework for partitioning and execution of data streaming application in mobile cloud computing. The framework contains the application modeling, architectural design and algorithm design. We provided a component based programming model to allow developers programming data application. We proposed a genetic algorithm to optimize the throughput in the partitioning of applications. At the cloud side, we proposed multi-tenancy Component-as-a-Service (CaaS) to achieve efficient utilization of cloud fabric resources. We demonstrated our proposed algorithm through simulation and one real world application, namely, QR-code recognition. The results show that

the partitioned execution with our proposed algorithm can achieve 2X throughput than the execution without partitioning.

With respect to the second problem, we proposed a framework for computation partitioning under dynamic mobile cloud environment. Based on the framework, we particularly studies the computation repartitioning problem under network fluctuations. We took a lot of measurements of the WiFi network bandwidth, and found that the bandwidth fluctuates extremely depending on the mobile user's location. We then exploited mobility prediction to estimate the user's future network quality, and designed an online algorithm to periodically update the partitioning during the application life cycle. The algorithm aims to minimize the total execution cost over the application life cycle. We collected network traces from our WiFi network testbed, and evaluated the online repartitioning algorithm with two applications. The results show that our online repartitioning algorithm can achieve at least 35% better performance than existing approaches under the real world network traces.

For multiple user computation partitioning, we focused on the model under which there exists a number of users offloading the computations onto a set of cloud servers. The users may compete for the resources on the cloud, so the offloaded computations can not be executed on the cloud immediately. We first illustrated that the Multi-user Computation Partitioning Problem (MCPP) is more difficult than traditional job scheduling problem in parallel computing. We then proposed a offline heuristic algorithm, namely *SearchAdjust*, to solve MCPP. Based on *SearchAdjust*, we further developed an online algorithm that can be deployed easily in practical systems. We evaluated the offline and online algorithm using the wikipedia traces, and showed through benchmarks that both algorithms have better performance than existing classical job scheduling algorithms in term of application delay.

In summary, computation partitioning is a useful technique to enhance the performance of mobile cloud applications. We identified several important and challenging problems in

computation partitioning under different models of application and system: computation offloading, single user computation partitioning, and multiple user computation partitioning. The evaluation results show that our approaches can increase both the end-user and system performances.

7.2 Future Research

We close this thesis by providing some suggestions for future research. Specifically, we believe that the following aspects are worth further investigations.

First, in Chapter 3, we study computation offloading for accurate and efficient RFID tracking. In the system model, we assume the period of collecting the RFID reading is fixed. It is interesting to involve the sampling period into the tracking algorithm design and the offloading decision. Another direction is to learn the RFID sensing model on the fly rather than through offline experiments. Furthermore, in this work, we focus on the two performance metrics: accuracy and algorithm running time. Energy consumption can be one important metric in many application and systems. It is interesting to study how the computation offloading can save energy for mobile devices.

Second, in Chapter 4, we develop a framework for partitioning and execution of data stream applications. The application contains a set of components. The partitioning problem is to decide which components are executed on mobile devices and which components are executed on the cloud. It is assumed that the component has one execution instance. However, in future work, we can consider that one component can have multiple instances as long as the input data of this component has parallelism and can be partitioned. The computation partitioning problem needs to decide the number of instances for each component, and the execution places for every instances. In Chapter 5, we study the computation

repartitioning in dynamic mobile cloud environment. To make it simple, we take the fluctuate network bandwidth as a case study. During the application life cycle, the workload of the device can also vary with time. In future work, we can study the computation repartitioning problem under both the device workload fluctuation and network bandwidth fluctuation.

Finally, in Chapter 6, we study multiple user computation partitioning problem when the mobile users complete for the computation resources on the cloud. The first direction for future research is to study the provisioning of computation offloading as a service to the end users from the standpoint of service provider. The economic cost on the cloud resource should be taken into account. Furthermore, we simplify that the users access to the cloud via separate wireless networks. In practical systems, the users may access to the cloud through the same wireless networks. The users can share and compete for the bandwidth resources. The allocation of network resources to the users should be considered together with the partitioning decision of each user. In future work, it is interesting to study the joint network resource allocation and computation partitioning problem.

Bibliography

[AAB+05] D. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *Proc. of CIDR*, pages 277–289. ACM Press, 2005.

[AK96] I. Ahmad and Y. Kwok. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multi-processors. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 7(5):506–521, 1996.

[AMGC02] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174C188, 2002.

[BB99] M. Bomze and M. Budinich. *Handbook of Combinatorial Optimization.* Springer, 1999.

[BB02] M. Bomze and M. Budinich. *Scheduling Theory, Algorithms, and Systems (2nd Edition).* Prentice Hall, 2002.

[BGGT07] J. Blumenthal, R. Grossmann, F. Golatowski, and D. Timmermann. Weighted centroid localization in zigbee-based sensor networks. In *Proc. of IEEE International Symposium on Intelligent Signal Processing*, pages 1–6, 2007.

[BHE00] N. Bulusu, J. Heidemann, and D. Estrin. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, 2000.

[BHE02] N. Bulusu, J. Heidemann, and D. Estrin. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[BHE08] N. Bulusu, J. Heidemann, and D. Estrin. Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and Mobile Computing*, 4(3):303–334, 2008.

[BHE10] N. Bulusu, J. Heidemann, and D. Estrin. Centroid localization of uncooperative nodes in wireless networks using a relative span weighting method. *EURASIP Journal on Wireless Communications and Networking*, 10(11):1–11, 2010.

[BKMS13] M. Barbera, S. Kosta, A. Mei, and J. Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *Proc. of INFOCOM*, pages 1285–1293, 2013.

[BSPO03] R. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics based remote execution for mobile computing. In *Proc. of MobiSys*, pages 945–953. ACM Press, 2003.

[BT08] R. Behnke and D. Timmermann. Awcl: Adaptive weighted centroid localization as an efficient improvement of coarse grained localization. In *Proc. of 5th Workshop on Positioning, Navigation and Communication*, pages 243–250, 2008.

[CBB+03] M. Cherniack, H. Balakrishnan, M. Balazinska, D Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Proc. of CIDR*, pages 422–440. ACM Press, 2003.

[CBC10] E. Cuervoy, A. Balasubramanianz, and D. Cho. Maui: Making smartphones last longer with code offload. In *Proc. of MobiSys*, pages 277–289. ACM Press, 2010.

[CCD+03] S. Chandrasekaran, O Cooper, A. Deshpande, M. Franklin, and et.al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proc. of SIGMOD*, pages 668–668. ACM Press, 2003.

[CIM+11] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys*, pages 301–314, 2011.

[DA98] S. Darbha and D. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 9(1):87–95, 1998.

[DG08] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *ACM Communication Magzine*, 51(1):107–113, 2008.

[FPS02] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proc. of ICDCS*, page 217, 2002.

[FWG10] Y. Fang, F. Wang, and J. Ge. A task scheduling algorithm based on load balancing in cloud computing. *Web Information System and Mining Lecture Notes in Computer Science*, 6318(1):271–277, 2010.

[GCWK12] P. Gao, A. Curtis, B. Wong, and S. Keshav. It is not easy being green. In *Proc. of SIGCOMM*, pages 211–222. ACM Press, 2012.

[GJM+12] M. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen. Cometcode offload by migrating execution transparently. In *Proc. of OSDI*, pages 93–106, 2012.

[GM10] E. Giampaolo and F. Martinelli. Robot localization by sparse and passive rfid tags. In *Proc. of IEEE International Symposium on Industrial Electronics*, page 1937C1942, 2010.

[GRJ+09] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proc. of Middleware*, pages 1–20. ACM Press, 2009.

[HBF+04] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and localization with rfid technology. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 1015–1021, 2004.

[HCL10] G. Huerta-Canepa and G. Lee. A virtual cloud computing provider for mobile devices. In *Proc. of MCS*, pages 3756–3761. ACM Press, 2010.

[HGSZ10] J. Hu, J. Gu, G. Sun, and T. Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Proc. of International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pages 89–96, 2010.

[HJG06] X. Huang, R. Janaswamy, and A. Ganz. Scout: Outdoor localization using active rfid technology. In *Proc. of BROADNETS*, pages 1–10, 2006.

[HJG07] X. Huang, R. Janaswamy, and A. Ganz. Rfid-based 3-d postioning schemes. In *Proc. of INFOCOM*, page 1235C1243, 2007.

[HLL07] S. Han, H. Lim, and J. Lee. An efficient localization scheme for a differential-drive mobile robot based on rfid system. *IEEE Transaction on Industrial Electronics*, 54(6):362C369, 2007.

[HPALP09] V. Hoonkavirta, T. Perala, S. Ali-Loytty, and R. Piche. A comparative survey of wlan location fingerprinting methods. In *Proc. of WPNC*, pages 243–251, 2009.

[HSW96] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: offline and on-line algorithms. In *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, 1996.

[IBY+07] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proc. of EuroSys*, pages 59–72. ACM Press, 2007.

[JPB09] D. Joho, C. Plagemann, and W. Burgard. Modeling rfid signal strength and tag detection for localization and mappingy. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 1015–1021, 2009.

[JSA00] P. Jacquet, W. Szpankowski, and I. Apostol. An universal predictor based on pattern matching, preliminary results. *Mathematics and Computer Science: Algorithms, Trees, Combinatorics and Probabilities*, pages 75–85, 2000.

[KAH+12] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in cloud for mobile code offloading. In *Proc. of INFOCOM*, pages 945–953, 2012.

[KL08] K. Kumar and Y. Lu. Cloud computing for mobile users: Can offloading computation save energy. *Computer*, 43(4):51–56, 2008.

[KLW+09] B. Kusy, H. Lee, M. Wicke, N. Milosavljevic, and L. Guibas. Predictive qos routing to mobile sinks in wireless sensor networks. In *Proc. of IPSN*, pages 109–120. ACM Press, 2009.

[KM09] K. Killourhy and R. Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Proc. of DSN*, pages 125–134, 2009.

[Kri09] M. Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *Proc. of PerCom*, pages 217–226, 2009.

[LB99] T. Lane and C. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[LBC98] T. Liu, P. Bahl, and I. Chlamtac. Mobility modeling, location tracking, and trajectory prediction in wireless atm networks. *IEEE Journal on Selected Area in Communications*, 16(6):922–936, 1998.

[LBC06] T. Liu, P. Bahl, and I. Chlamtac. Evaluating next-cell predictors with extensive wi-fi mobility data. *IEEE Transaction on Mobile Computing*, 5(12):1633–1649, 2006.

[LCS06] X. Liu, M. Corner, and P. Shenoy. Ferret: Rfid localization for pervasive multimedia. In *Proc. of UbiComp*, page 422C440, 2006.

[Low04] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[LW02] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(1):699–719, 2002.

[LWK+10] H. Lee, M. Wicke, B. Kusy, O. Gnawali, and L. Guibas. Data stashing: Energy-efficient information delivery to mobile sinks through trajectory prediction. In *Proc. of IPSN*, pages 291–302. ACM Press, 2010.

[LWX09] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[LZ08] Y. Lee and A. Zomaya. A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(9):1215–1223, 2008.

[LZ11] Y. Lee and A. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(8):1374–1381, 2011.

[Mar09] E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. In *Master Thesis, Carnegie Mellon Universtiy*, pages 277–289, 2009.

[MSY12] S. Maguluri, R. Srikant, and L. Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proc. of INFOCOM*, pages 702–710, 2012.

[NKSH09] S. Nousiainen, J. Kilpi, P. Silvonen, and M. Hiirsalmi. Anomaly detection from server log data: A case study. *VTT Tiedotteita Research Notes*, 2480, 2009.

[NLLP03] L. Ni, Y. Liu, Y. Lau, and A. Patil. Landmarc: indoor location sensing using active rfid. In *Proc. of PerCom*, page 407C415, 2003.

[PH09] S. Park and S. Hashimoto. Autonomous mobile robot navigation using passive rfid in indoor environment. *IEEE Transactions on Industrial Electronics*, 7(56):2366C2373, 2009.

[PM10] J. Paul Morrision. *Flow-Based Programming: A New Approach to Application Development (2nd Edition)*. CreateSpace, 2010.

[Raj91] J. Raj. The art of computer systems performance analysis: Techniques for experimental design, meas-urement, simulation, and modeling. *Wiley-Interscience*, 1991.

[RRA08] J. Rellermeyer, O. Riva, and G. Alonso. Alfredo: An architecture for flexible interaction with electronic devices. In *Proc. of Middleware*, pages 22–41. ACM Press, 2008.

[RSM+11] M. Ra, A. Sheth, L. Mummert, P. Pillai, and D. Wetherall. Odessa: enabling interactive perception applications on mobile devices. In *Proc. of MobiSys*, pages 43–56. ACM Press, 2011.

[SAZN12] C. Shi, M. Ammar, E. Zegura, and M. Naik. Computing in cirrus clouds: The challenge of intermittent connectivity. In *Proc. of MCC*, pages 23–28. ACM Press, 2012.

[SBCD09] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 6(4):12–23, 2009.

[SR02] S. Seiden and B. Rouge. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

[SSSS11] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Proc. of ICDCS*, pages 559–570, 2011.

[SV07] S. Schneegans and P. Vorst. Using rfid snapshots for mobile robot self-localization. In *Proc. of European Conference on Mobile Robots*, pages 1–6, 2007.

[TFBD00] S. Thrun, D. Fox, W. Bugard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99–141, 2000.

[THW02] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 13(3):260–274, 2002.

[UBG04] E. Uysal-Biyikoglu and A. Gamal. On adaptive transmission for energy efficiency in wireless data networks. *IEEE Transaction on Information Theory*, 50(12):3081–3094, 2004.

[UPS09] G. Urdaneta, G. Pierre, and M. Steen. Wikipedia workload analysis for decentralized hosting. *Operations Research*, 53(11):1830–1845, 2009.

[VZ08] P. Vorst and A. Zell. Semi-autonomous learning of an rfid sensor model for mobile robot self-localization. In *Proc. of European Robotics Symposium*, page 273C282, 2008.

[WGKN08] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. Using bandwidth data to make computation offloading decisions. In *Proc. of IPDPS*, pages 1–8, 2008.

[WLC12] F. Wang, J. Liu, and M. Chen. Calms: Cloud-assised live media streaming for globalized demands with time/region diversities. In *Proc. of INFOCOM*, pages 199–207, 2012.

[WLZ+12] Y. Wu, B. Li, L. Zhang, Z. Li, and F. Lau. Scaling social media applications into geo-distributed clouds. In *Proc. of INFOCOM*, pages 211–222, 2012.

[YCCJ13] L. Yang, J. Cao, H. Cheng, and Y. Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. In *Technical Report*. Dept. of Computing, Hong Kong Polytechnic University, 2013.

[YCT+12] L. Yang, J. Cao, S. Tang, T. Li, and A. Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. In *Proc. of CLOUD*, pages 794–802, 2012.

[YCT+13] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri. Foreseer: Predictive mobile-cloud program partitioning under network fluctuations. In *Technical Report*. Dept. of Computing, Hong Kong Polytechnic University, 2013.

[YCY+13] L. Yang, 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.

[YOC08] K. Yang, S. Ou, and H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communication Magazine*, 46(1):56–63, 2008.

[YTH04] K. Yamano, K. Tanaka, and M. Hirayama. Self-localization of mobile robots with rfid system by using support vector machine. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 3756C3761, 2004.

[ZC04] Y. Zou and K. Chakrabarty. Sensor deployment and target localization in distributed sensor networks. *IEEE Transaction on Embedded Computing and System*, 3(1):61–91, 2004.

[ZKJG09] X. Zhang, 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):379–394, 2009.