Design and Optimization of Mobile Cloud Computing Systems with Networked Virtual Platforms

YoungHoon Jung

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

©2016

YoungHoon Jung

All Rights Reserved

ABSTRACT

Design and Optimization of Mobile Cloud Computing Systems with Networked Virtual Platforms

YoungHoon Jung

A *Mobile Cloud Computing* (MCC) system is a cloud-based system that is accessed by the users through their own mobile devices. MCC systems are emerging as the product of two technology trends: 1) the migration of personal computing from desktop to mobile devices and 2) the growing integration of large-scale computing environments into cloud systems. Designers are developing a variety of new mobile cloud computing systems. Each of these systems is developed with different goals and under the influence of different design constraints, such as high network latency or limited energy supply.

The current MCC systems rely heavily on *Computation Offloading*, which however incurs new problems such as scalability of the cloud, privacy concerns due to storing personal information on the cloud, and high energy consumption on the cloud data centers. In this dissertation, I address these problems by exploring different options in the distribution of computation across different computing nodes in MCC systems. My thesis is that "*the use of design and simulation tools optimized for design space exploration of the MCC systems is the key to optimize the distribution of computation in MCC.*"

For a quantitative analysis of mobile cloud computing systems through design space exploration, I have developed NETSHIP, the first generation of an innovative design and simulation tool, that offers large scalability and heterogeneity support. With this tool system designers and software programmers can efficiently develop, optimize, and validate large-scale, heterogeneous MCC systems. I have enhanced NETSHIP to support the development of ever-evolving MCC applications with a variety of emerging needs including the fast simulation of new devices, e.g., Internet-of-Things devices, and accelerators, e.g., mobile GPUs.

Leveraging NETSHIP, I developed three new MCC systems where I applied three variations of a

new computation distributing technique, called *Reverse Offloading*. By more actively leveraging the computational power on mobile devices, the MCC systems can reduce the total execution times, the burden of concentrated computations on the cloud, and the privacy concerns about storing personal information available in the cloud. This approach also creates opportunities for new services by utilizing the information available on the mobile device instead of accessing the cloud.

Throughout my research I have enabled the design optimization of mobile applications and cloud-computing platforms. In particular, my design tool for MCC systems becomes a vehicle to optimize not only the performance but also the energy dissipation, an aspect of critical importance for any computing system.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I thank my committee members who were more than generous with their expertise and precious time. Thank you Prof. Alfred V. Aho, Prof. Martha Kim, Prof. Li E. Li, and Dr. Ethan Kim for agreeing to serve on my committee. A special thanks to Prof. Luca P. Carloni, my advisor, for his countless hours of reflecting, reading, supporting, encouraging, and most of all being a great mentor throughout the process.

I would like to acknowledge and thank my friends and colleagues in System-Level Design Group, including Dr. Richard Neill, Dr. Michele Petracca, Youngjin Yoon, Dr. Marcin Szczodrak, Dr. Hung-Yi Liu, Emilio Cota, Paolo Mantovani, Dr. Johnnie Chan, Dr. Nicola Concer, Dr. Guiseppe D. Guglielmo, Dr. Christian Pilato, and Jihye Kwon for helping me with ideas, advices, and feedbacks. My appreciation goes to Karl Stratos, Jaehwan Koo, and Jinhyung Park, my dear friends who collaborated on the projects. Their excitement and willingness to cooperate made the completion of this research an enjoyable experience.

Appreciation is extended to my old colleagues and friends who finished their doctoral programs during my study or are still working on them, including Dr. Changewoo Min, Dr. Suinn Park, and Byoungyoung Lee. They have inspired me throughout the study. Thank you Dr. Hyunwoong Shin for suggesting me to start a doctoral study in the first place. I would deeply appreciate Prof. Jaeyoung Choi, Prof. Soowon Lee, and Prof. Junbeom Yoo for their support with advice and concerns when I decided to begin the study.

Many thanks to my parents for always believing in me even at a distance of 6,800 miles. Finally, I would like to express my appreciation to Mihyun Kim, my wife, for being a psychological companion and helping me focus on the research by taking good care of our lovely kids during the entire process as always.

To my loving parents, family and many friends.

Chapter 1

Introduction

An increasing number of computing systems rely on a set of backend services operated on cloud computers while providing their primary user interfaces on mobile devices [30; 113; 149]. This new class of emerging computing systems, commonly termed *Mobile Cloud Computing* (MCC) systems, has gained growing popularity in many application domains such as e-commerce [205; 62], learning [248; 194], healthcare [195; 355], gaming [334; 331], social networks [335; 225], and so on. Behind the popularity of MCCs across various domains is the blossoming of two technologies: the wide use of cloud computing and the explosive growth of mobile devices. First, cloud computing has increasingly become the standard way to operate Internet-based services, preferred by the service providers due to its low prices, high performance, and flexibility. These have been the main driving forces that increased the instances of cloud servers built in data centers. The number of data centers being built around the world is expected to continue to increase until 2017 when it will peak at 8.6 million [150]. The estimated total space for data centers will reach 1.94 billion square feet in 2018. Second, more and more users access cloud services through their mobile devices which can provide a richer user experience, i.e., easy-to-use, intuitive, and interactive, user interfaces than using personal computers [112].

By combining these technologies, MCCs offer many advantages and enable new services. For instance, an MCC speech recognition application takes a segment of the user's voice and sends it to the cloud. The cloud processes the segment and sends the recognition result back to the mobile user. This approach has various advantages compared to running the algorithm on the mobile. First of all, harnessing the powerful processing cores on the cloud allows fast execution of the speech

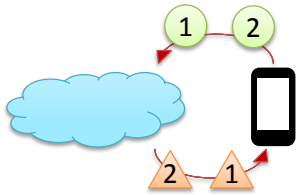


Figure 1.1: Computation Offloading.

recognition algorithm. Additionally, the database necessary to execute the speech recognition algorithm is stored on the cloud servers, thus freeing space from the limited storage of the mobile device. Meanwhile, the mobile user interface provides a simple yet convenient and ubiquitous way for the users to interact with the application. Lastly, the reduced use of the processing power on the mobile greatly contributes to saving the limited energy budget.

This particular form of task delegation from the mobile to the cloud is called *Computation Offloading*, or simply *Offloading* [208]. Due to its benefits, offloading is very frequently adopted in MCC applications and frameworks. For instance, computation offloading can reduce the mobile network traffic [87], decrease the cloud resource provision cost [286], or reduce the execution time and energy consumption for running mobile applications [333]. Figure 1.1 illustrates a mobile device offloading two tasks (Task 1 and Task 2) to the cloud and receiving the corresponding results back from the cloud (Result 1 and Result 2). In this figure, a circle represents a task and a triangle is the corresponding result. A variety of efficient offloading techniques have been studied. These techniques are, however, mainly focused on the efficiency of the mobile. For example, some offloading techniques aim to achieve faster total execution time [327], less energy consumption on the mobile [66; 111], or both [72; 111]. Eventually, the large amount of offloading translates into more frequent use of the cloud and increased amount of computation from the cloud's perspective. These increased needs have started to overburden the cloud, thus creating new problems or worsen existing cloud issues.

- The cloud computing services which operate on the data centers currently suffer from limited scalability. It is difficult to increase the number of server computers hosted in one data center beyond a certain level. Moreover, finding locations and funds to build new data centers is challenging. Behind these challenges, there are some technical constraints like heat management, network, or power supply [126; 127; 332; 65] and social issues such as concerns for the

impact on residential environments [307].

- As a growing amount of personal data is stored on the cloud, users are increasingly concerned about privacy issues. Although it is uncertain whether it is safer to store personal data on the cloud or on the mobile, having the cloud as a data backup, i.e., storing the data on both the mobile and the cloud, might increase the probability that it can be illegally accessed. The recent leaks of celebrity photos in 2014 [305; 306] is proof that the security issue on the cloud system can turn into a large-scale threat for personal privacy.

- The energy consumed by data centers is enormous. Data centers are the largest and fastest growing electricity consumer. US data centers consumed approximately 91 billion kilowatt-hours of electricity in 2013. This amount was twice as much as the power consumed by all the households in New York City the same year. The energy consumption by data centers is anticipated to reach 140 billion kilowatt-hours by 2020 [52]. The companies that operate these large-scale data centers are under pressure to reduce their energy consumption from governments and environmental organizations.

The fundamental research question behind these problems is **how to optimally distribute computation across different computing nodes** in the systems. The question is very important particularly for MCC systems where two very distinct types of computers must cooperate. For instance, preprocessing the data on the mobile device before transferring them to the cloud may substantially decrease the amount of data the cloud must process as well as the network use. This is a well-known question studied by many researchers who have proposed different solutions optimized to achieve different objectives. Examples include offloading for execution times based on resource monitoring [347], migrating tasks to the cloud for saving energy [196], distributing application binaries automatically [146], and partitioning the application dynamically [69].

However, these approaches are not suitable for the increasing number of MCC applications due to the following limitations:

- System optimization is application specific. Different applications benefit from different heuristics that can largely save the total amount of computation required by the applications. An automated analysis cannot invent a new technique to optimize the application. Instead,

a system designer with a fast and effective design tool is more likely to come up with an efficient optimization idea through design-space exploration.

- Existing methods cannot support heterogeneity and very large scalability, the two most distinguishing characteristics of MCC systems. MCC systems are heterogeneous because they consist of cloud computers and mobile devices. Meanwhile, those mobile devices have varying computing processors and networks. In addition, recent large-scale MCC systems serve more than a billion users with more than tens of millions of them accessing the cloud at the same moment.

- Unlike mathematical optimization problems, where the optimal solution can be calculated by problem solving algorithms, system optimization can often be solved only with a trial-and-error approach. This is due to the different granularity of resource units and the little-known behavior of certain resources when the degree of their use changes, e.g., congestion in the network. A number of other elements each of which is too minor to be modeled into the objective function are also important as the outcome can significantly vary by the collective changes of those elements. The trial-and-error method can take too much time in system optimization without using a well-designed tool that is optimized for the given domain.

- Computation offloading is a viable model only under the circumstance where the growth of cloud computers exceeds the growth of mobile devices. Therefore, offloading is not applicable to many MCC systems where mobile devices may outgrow the available cloud resources. Furthermore, computation offloading is increasingly used also in other types of fast-growing computing systems such as the Internet of Things [123] and Cloud Robotics [177]. Hence, the computational burden on the cloud is expected to intensify further in the near future.

The thesis presented in this dissertation is "*the use of design and simulation tools optimized for design space exploration of the MCC systems is the key to optimize the distribution of computation in MCC.*" In this dissertation, I study the optimal distribution of computation in the MCC systems by

1. *developing effective and efficient design and simulation tools to support the design and optimization of large-scale heterogeneous MCC systems* and

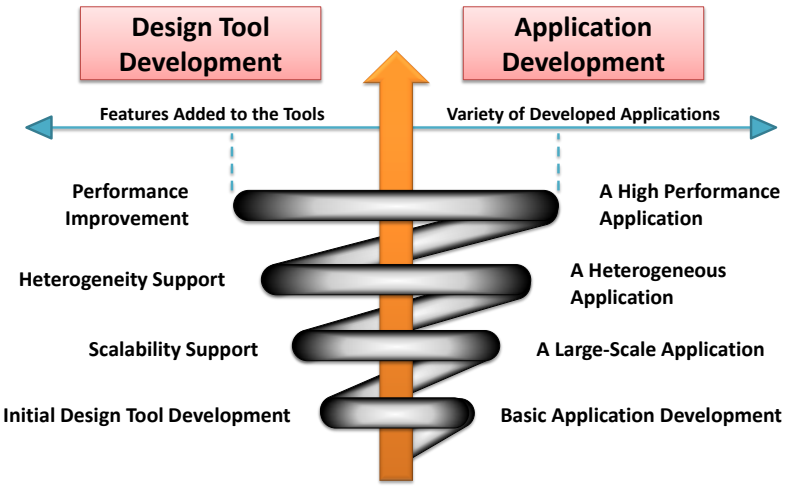


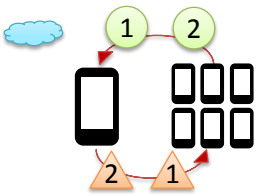
Figure 1.2: A cooperative development process of design tools and applications.

2. *inventing a class of optimization patterns for computation distribution and apply them to the new MCC systems I develop using my tools.*

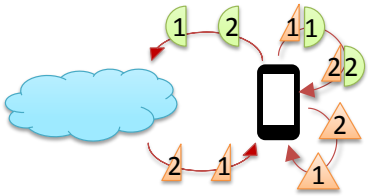
1.1 The Iterative Development Process

The way I investigate the distribution of computation for MCC systems in this thesis is an iterative process of developing design tools and implementing MCC systems where distinct distributions of computation can be applied. The development of design tools and applications can benefit each other, accelerating the completion of both [136]. As shown in Figure 1.2, these two development processes are iteratively connected, resulting in one cooperative process. The orange-colored vertical arrow at the center of the figure indicates the time spent on the development. As the development project progresses, the level of the development completion, or the project maturity, increases. The project maturity can be measured by the number of features added to the developed tools and the variety of the developed applications.

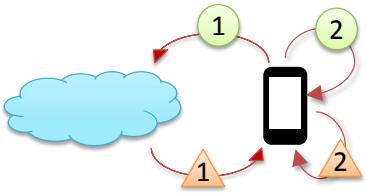
A good design tool can make the development and testing of a new application more efficient [308; 158]. For instance, a design tool that supports great scalability for simulating many nodes can effectively and efficiently assist the developers of a new type of applications that run on a large number of distributed mobile devices, thus reducing the time for deploying and testing on the many physical devices and decreasing time-to-market. Likewise, when the application runs on heterogeneous cores, a tool that supports these distinct types of cores will be very useful. Without



(a) Reverse Distributed Offloading



(b) Algorithm-Division Reverse Offloading



(c) Query-Division Reverse Offloading

Figure 1.3: A comparison of offloading and three distinct reverse offloading techniques.

an appropriate design tool, application development can be slow.

On the other hand, without knowing actual applications, the tools cannot be enhanced in a meaningful direction. In particular, the feedback information obtained from application development and test usage of the tool is the key source to improving the tool [98]. Therefore, the processes of developing design tools and applications are mutually beneficial. A close collaboration between the improvement of a design tool and the development of applications based on the tool can speedup the process and refine the quality of the tool as well as the developed applications.

1.2 Reverse Offloading

Throughout the development of various MCC applications, I invented a series of techniques that I group under the term *Reverse Offloading*. Reverse offloading distributes computation across the cloud and the mobile devices, instead of offloading the entire tasks to the cloud. Figure 1.3(a) - 1.3(c) illustrates three different types of reverse offloading. First, *Reverse Distributed Offloading* distributes the entire tasks across multiple mobile devices in the same network. This makes the mobile independent from the cloud, as shown in Figure 1.3(a). In other words, the cluster formed out of the mobile devices works like an alternative cloud. Second, *Algorithm-Division Reverse*

Offloading splits each task into two portions. As shown in Figure 1.3(b), the application sends a portion of the task to the cloud; then, the intermediate result for the portion of the task returns to the mobile; finally, the mobile device runs the rest of the task taking as input the intermediate result returned from the cloud. Third, *Query-Division Reverse Offloading* selectively sends only some of the tasks to the cloud. I discuss these three techniques in the context of three MCC applications in Chapters 7, 8, and 9, respectively.

Reverse offloading can effectively address the aforementioned challenges as follows:

- It reduces the amount of computation that the cloud must run. Reverse distributed offloading makes the mobile devices independent from the cloud. In other words, the computational burden on the cloud is dissolved. MCC applications that adapt algorithm-division reverse offloading will only offload certain portions of the tasks to the cloud. In this case, the rest is saved from offloading. Query-division reverse offloading sends only certain tasks to the cloud. Likewise, the tasks handled by the mobile devices are saved from offloading. Particularly, the amount of computation reduced by reverse offloading is proportional to the number of mobile devices, which is the most rapidly growing factor in the MCC systems. The fast improving computational power and energy efficiency of mobile devices also contribute to the effectiveness of reverse offloading.

- Reverse offloading can effectively decrease the total execution time for some applications.This is due to the reduced use of the network, particularly on the mobile side. The mobile network, i.e., Wi-Fi or LTE, takes a large portion of the execution time and the energy consumption of MCC applications. By reducing the network use, reverse offloading inherently decreases the network latency and energy dissipation. Also, the growing processing power on mobile devices, e.g., mobile CPUs and GPUs, contributes significantly to the feasibility of reverse offloading.

- A certain type of reverse offloading can mitigate some privacy concerns. Instead of sending personal information to the cloud (while also leaving it on the mobile), securing it only on the mobile provides less chance for the personal information to be leaked.

- Reverse offloading can enable new services. For instance, in Chapter 8 I introduce an MCC application that uses personal information in the users' emails by extracting the information

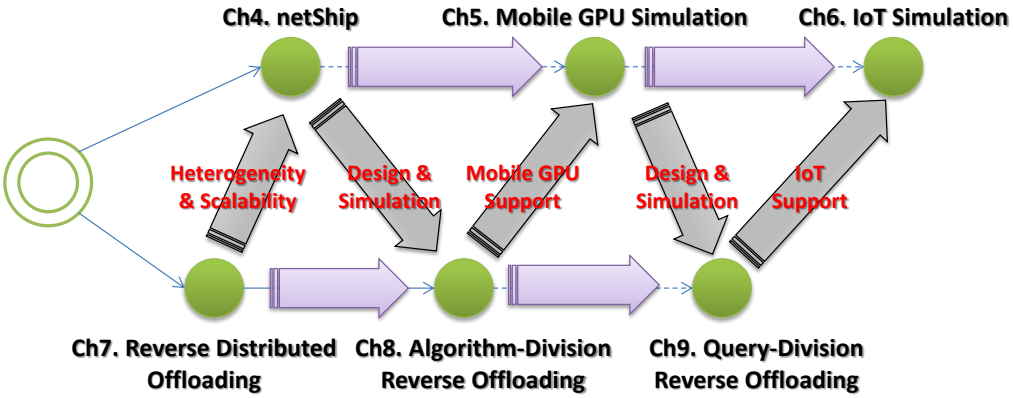


Figure 1.4: The co-development process of MCC applications and tools.

from the messages stored on the mobile devices. So far this type of service has been possible only for a small group of large companies that offer their own email services, e.g., Google, Microsoft, and Yahoo. Instead of accessing email services on the cloud, leveraging the personal information in the email stored on the mobile makes a variety of new services feasible for many small independent companies.

1.3 Outline of the Dissertation

The rest of the thesis is organized as follows. In the first two chapters, I introduce two important concepts. First, in Chapter 2, I describe mobile cloud computing (MCC), which is currently one of the most popular forms of distributed computing architecture. It is a combination of of mobile computing and cloud computing, which provides both mobility and flexibility. Chapter 3 is about virtual platforms (VP), a simulation model that enables hardware prototyping, early software development, easy deployment, and convenient test. Since the goal of this thesis includes developing MCC applications using virtual platform-based design tools, these two concepts are fundamental for the rest of the thesis.

In the rest of the thesis, I introduce the design tools and MCC applications I developed and how each of them contributes to one another as shown in Figure 1.4. In particular, across Part I and II each chapter describes a particular technical contribution as well as the way it becomes a stepping stone for the next project. In Part I (through Chapter 4 to 6), I introduce the design tool I invent

for simulating MCC systems. In Chapter 4, I present NETSHIP, a design tool for developing MCC applications. This tool supports the large scalability and heterogeneity required by the MCC systems. NETSHIP was presented at DAC'13 [167]. At a later stage, I enhanced NETSHIP to support the development and simulation of MCC applications that use the mobile Graphic Processing Units (GPUs) which are present on the mobile devices. The enhanced simulation framework multiplexes the host GPU and estimates execution time and power consumption of the GPU code on the mobile GPUs, as described in Chapter 5. This framework, including the estimation techniques, was presented at DAC'15 [164]. In Chapter 6, I discuss how to further augment NETSHIP to simulate IoT-based systems and present some preliminary results on this enhancement.

In Part II (through Chapter 7 to 9), I introduce the MCC systems I developed using my design tool. In Chapter 7, I introduce the reverse distributed offloading adapted on a cluster of embedded systems. The material in this chapter was published at CloudCom'12 [166]. In Chapter 8, I introduce an MCC application where algorithm-division reverse offloading is applied. Specifically, this study shows that the execution can be faster when the mobile processing core is more actively used, mainly due to the reduced amount of network use by reverse offloading. The material in this section was presented at WWW'15 [169]. In Chapter 9, I describe Query-Division Reverse Offloading and an MCC application where the Query-Division Reverse Offloading is applied. This application is an audio stream retrieval system where the client periodically send queries to the server. Query-Division Reverse Offloading significantly reduces the number of queries sent to the server by caching portions of audio database on the local device and allowing the local audio retrieval from the cached database.

In Part III (through Chapter 10 to 12), I describe the co-development process of the design tools and the MCC applications and I conclude my dissertation by offering my perspective on the future of MCC tools and applications.

Chapter 2

Mobile Cloud Computing Systems

Since the advent of the first computers, computer systems have advanced pushing their boundaries in multiple dimensions. Figure 2.1 shows three different dimensions of computing systems' development: computational capability, flexibility, and mobility. This figure is a simplified illustration that captures the three most important dimensions that will be discussed in this chapter.

Computational Capability. The users' ever-increasing needs for computations have made computer designers seek a way to process computations more efficiently, by increasing the capability of computing systems. The efforts for creating more efficient and powerful computing systems have created many new technologies such as faster computing cores, multi-core and multi-processor computers, and distributed computing systems. One of the most obvious ways to achieve a higher efficiency in computing systems is to make more powerful processing cores with a faster computation speed. The processing power of the computing cores has dramatically increased based on the increasing number of transistors that can be integrated on a chip. The growth rate in the number of integrated transistors has been close to exponential, approximately following Moore's Law [228]. Around 2005, the growth based on a single integrated computing processor reached a saturation wall where no single further-integrated processor is expected to keep up with the law [339]. This is due to various technical limitations that are difficult to improve within a single processing core, such as heat management, energy efficiency, or cross-talk of the designed chips. Multi-core and multi-processor computers overcome this issue by executing software in a parallel fashion on the combined multiple cores in each processor and multiple processors in a system [50]. The parallel use of multiple cores and processors is now considered critical for the continuation of Moore's

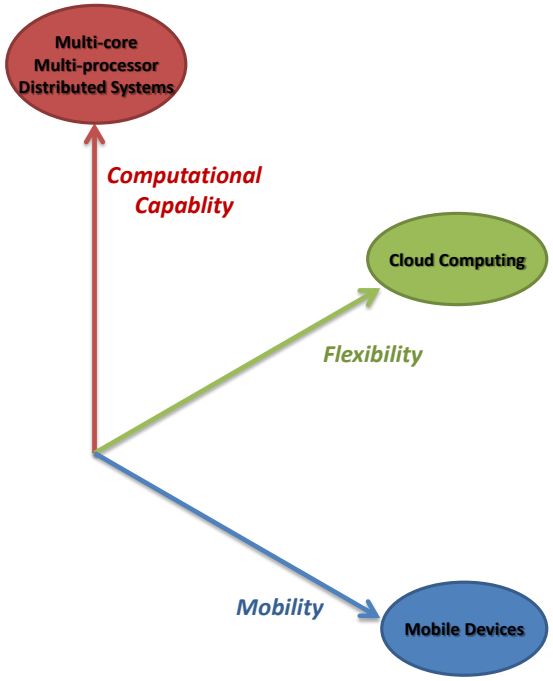


Figure 2.1: The three dimensional space in the advance of computing systems.

Law. There exist, however, some physical challenges that limit the number of processing cores and processors added to a single computer machine. Heat and power consumption, again, limit the number of cores that can be packed into a single processor. Although there exist some experimental computers with several thousands of computing processors, a usual and efficient way to build a highly parallel computing system is connecting multiple distributed server computers with a communication network. Therefore, forming a cluster of multiple computers is a way of providing more computational capabilities to satisfy the growing volume of user demands.

There are some open issues that prevent current computing systems from being further improved in terms of computational capabilities. A common issue that occurs in both multi-core and multi-processor computers and distributed computing systems is the communication overhead. To run a single application on these distributed computing cores, communications among them is necessary. In some cases, the overheads, e.g., communication time or energy spent for communication, exceed the advantages of using more computational elements. When this happens, it might be desirable to have fewer computing elements in the system unless the distribution of computation could be improved to reduce the communicational overhead. Meanwhile, as Amdahl's law suggests, once the parallelizable portion of the work is sufficiently sped-up, there is not much additional gain from

adding even more processing cores to the system. Thus, to achieve high execution performance on the distributed multi-core computers, it is important to devise algorithms that can be efficiently parallelized. Improving the computational capabilities of the computing systems is one of the oldest, but still actively studied area, in computer science.

Flexibility. The *virtualization* technology was originally developed to run a virtual (as opposed to 'physically existing') computer on a host computer with a different machine architecture. This technique is useful developing a new type of computer or testing software for a new computer architecture without physically possessing the new computer at hand. Since the beginning of 21st century, a new type of services has emerged that use virtualization in a new way. These services offer a virtualized computing environment to their customers on an on-demand basis. Since the provided computing environment is virtualized, computer resources are used when the customers need them (and pay) whereas they could be returned to the service provider when the customers do not need them (and do not pay). This flexible computing system service is called *cloud computing* and the payment system for cloud computing is called *pay-as-you-go* as the customers incur the service charge only when they use the service. This flexibility of cloud computing has created a number of advantages for both the cloud providers and their customers. First, the customers are free from buying, managing, fixing, upgrading, emergency-caring, and expanding the physical computing infrastructure they use. Cloud computing provides a very easy-to-use interface to start a new cloud instance instantly. The repair of broken machines, physical system upgrade, or migration on emergency cases are the responsibility of the cloud service providers, who have more expertise on these troublesome works. The cloud customers can focus more on their own businesses without being distracted by operating their own computing servers. Second, from a business point of view, the pay-as-you-go system saves the upfront cost when you start a new business since there is no need to purchase the whole computing infrastructure including the server computers, power management devices, or the network equipments. This makes it easy for many companies to adapt cloud computing for their newly launched businesses. Third, the service providers do not have to over-provision computing instance as if all their customers access the service at the same time, since the customers need the service from time to time. Instead, a smaller size of computing infrastructure that can support the usual usage is enough. During the peak time, computer providers can dynamically increase the computing resources to serve the users. This greatly saves the costs for the service providers.

On the other hand, cloud computing has been facing some newly introduced challenges. For instance, the integrated data centers result in an intensive use of electricity. This leads to several problems such as electric outage, increased electricity costs, and pollution from electricity generation.

Mobility. Mobile phones have been a must-have item for people living in today's world. The wide use of smartphones changed the way people live. Beside using the phones the way it used be, i.e., calling and corresponding text messages, people now take pictures with their phones, find directions to the places they visit for the first time using the map service, watch video clips, play video games in their space time, and so on. All these activities rely on the phones' computational abilities as a small 'computer'. As a new instance of modern computing, mobile devices have all the characteristics of the computing systems, having central processing units, data buses, main memory, storages, and I/O modules. On top of these, they have some additional requirements as a special form of computers. They need mobility, which also requires a portable power source, e.g., batteries, wireless network connectivity to access the services beyond the offline contents, a small size enough to be easily carried, e.g., in a pocket, and good heat management to be contained in a relatively small sized case.

Mobile computing is one of the most rapidly developing area of the computing system market in both quantity and quality. The number of mobile device exceeds 8 billion in 2014 and is expected to continue to grow over 12 billion by 2018 [258]. During the same period, the number of transistors used in the mobile processing cores is expected to grow up by 10 times [48]. The computing ability of the mobile devices these days has been growing so quickly that it is comparable to what supercomputers could do two decades ago.

Since these dimensions are orthogonal, a computing system can be expanded toward one or more dimensions at the same time. For instance, most cloud computing systems are already designed for high-performance, consisting of a cluster of multiple distributed computers with many multi-core processors. Also, many recent mobile devices are equipped with a multi-core processor along with high-performing mobile GPUs. Another combination of the other two dimensions is the integration of cloud computing and mobile computing. The new type of computing systems, called *Mobile Cloud Computing* (MCC) systems, is designed to offer the mobility and rich experience of the mobile device while the cloud computers that back up the system handle the high demands for the

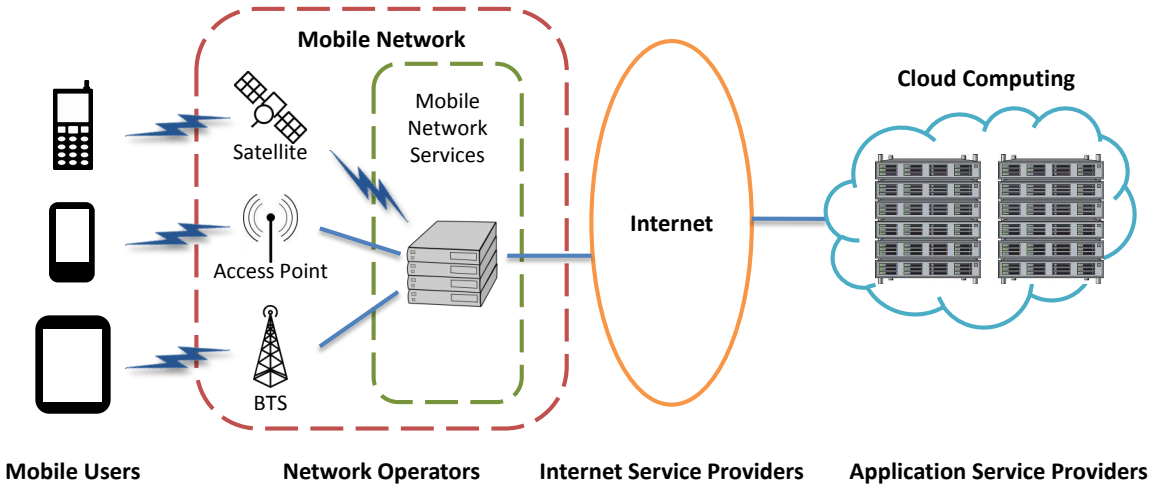


Figure 2.2: The architecture of Mobile Cloud Computing systems.

computation-heavy tasks along with the access to the infinite-sized cloud storage.

Next, I compare the different definitions of MCC and discuss the advantages of MCC. Thereafter, I introduce some examples of MCC systems and applications and discuss the current issues in the development of MCC systems.

2.1 Definitions

As shown in Figure 2.2, MCC refers to the amalgamation of mobile computing and cloud computing. The mobile users access the application services that reside in the cloud through the mobile network to which their devices are connected along with the Internet. Given the high popularity of the two technologies and their distinct advantages and disadvantages, the emergence of this combination is natural. Particularly, the portability and reachability of the mobile devices are two of the most significant advances, while their limited computational ability, storage size, and energy budget still need to be resolved. Although in its essence an MCC system can be simply explained as a computing environment where mobile devices and clouds are integrated, there are a few different variations to define MCC depending on their distinct perspectives.

- According to Liu et al, MCC is a model for the elastic augmentation of mobile device capabilities via ubiquitous wireless access to cloud storage and computing resources, with context-

aware dynamic adaption to changes in the operating environment [208].

- Sanaei et al. define MCC as a rich mobile computing technology that leverages unified elastic resources of various clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet, regardless of heterogeneous environments and platforms, based on the pay-as-you-use principle [272].

- Abolfazli et al. define an important concept in MCC, *Mobile Computation Augmentation*, as the process of increasing, enhancing, and optimizing computing capabilities of mobile devices by leveraging various feasible approaches, hardware and software [30; 31].

- Miluzzo et al. envision MCC systems as distributed computing architectures where mobile devices become core computing nodes due to their rapidly growing capabilities. They expect that heterogeneous devices, such as personal computers and set-top boxes, will be also engaged in the MCC infrastructure [221].

- Some researchers expect the emergence of a mid tier in MCC, called cloudlet [41]. A cloudlet is a trusted, resource-rich computer, or cluster of computers, that is well connected to the Internet and is available for use by nearby mobile devices. Thus, MCC becomes the seamless integration of cloudlet and public cloud with infrastructure specialization for mobile applications.

Overall, MCC is an integrated computing system of mobile devices and cloud computers where the users experience the advantages of mobile devices in terms of rich applications and mobility while the limited resources of mobile devices, e.g., the limited energy budget, relatively slower computational speed, and constraint in the storage size, are *augmented* by the cloud computers with their higher computational capabilities and larger storage capacities. This augmentation which is based on the delegating a portion of computation from mobile devices to cloud servers, is called in different ways, including computation offloading, task migration, and remote processing.

2.2 Advantages of MCC

Recent market surveys show that an increasing number of businesses are adopting the MCC architecture for their service infrastructure and that it is expected to exceed 240 million businesses by 2015 [285]. In MCC, the mobile devices are augmented by *offloading* heavy computations and large file objects to the cloud. This brings many advantages and is the main reason of the success of MCC. The major advantages of using MCC compared to non-cloud services that run only on the mobile devices are the following:

Prolonged battery lifetime. By migrating heavy computations and complex processing to cloud servers, mobile devices can save a significant amount of energy [280; 115].

Enhanced capabilities on mobile devices. By leveraging the stronger computational power and the larger data storage on cloud machines, the users can benefit from fast executions of computations and large data storage [348; 322].

High reliability and availability. By keeping data and applications in the clouds, the users can securely store them. Those data and applications are kept in a reliable way even when the mobile devices are broken, stolen, moved, or suffering from a low battery. The cloud service offers high-availability through their reliable and resilient architecture [226].

Flexible and transparent scalability. Cloud service provisioning is *elastic*, which means it is dynamically offered on an on-demand basis. Therefore, the mobile device users are not necessarily aware of the computation and storage offloading to the cloud even when their use increases [54].

2.3 MCC Systems and Applications

Thanks to its distinct advantages in various aspects, MCC has been adopted in functionally diverse areas. The application areas vary from finance to entertainment. Correspondingly, they are supported by a variety of different forms of system architecture. Many of these applications are an extension of existing online services where mobile devices are a convenient user interface tool at hand while keeping the wealthy computing environment offered by the cloud servers.

Mobile Commerce. Mobile commerce, or M-commerce, is the delivery of electronic commerce services directly into the customer's hand, anywhere, via wireless technology [205; 62]. Mobile commerce is a very wide area that includes mobile money transfer, mobile ATM, mobile

ticking, mobile coupons, mobile shopping, and mobile advertising. By integrating with the cloud, mobile commerce enhances its security and achieves ubiquitous, synchronized services across the globe. The market size of mobile commerce is $230 billion and is expected to reach $700 billion in 2017 [79].

Mobile Learning. E-learning, or online learning, has been transformed into mobile learning. At the early stage of M-learning, the services suffered from the high cost of devices and the high system performance requirements for playing videos. The continued growth of computational power in the mobile devices along with the integration with the cloud services have resolved this issue [248; 194]. Nowadays, students can easily find teachers and access the learning contents by using their mobile devices, spurring the success of mobile learning.

Mobile Healthcare. Healthcare is an application area where mobility is most appreciated. Whenever an emergency case happens, it is easy to reach mobile devices to ask for help. Mobile devices are also very convenient to keep close to the patients for 24-hour monitoring and checking up their medical records [195; 355].

Mobile Gaming. Mobile game is a fast increasing market with high revenue. Multiplayer games allow the gamers to interact among each other in the cloud, thus making the game more dynamic and communicative. Some games offload a portion or the entire game engine that requires heavy computations, e.g., graphic rendering, to the server in the cloud. Offloading can also save energy and thus increase the time users can play the game [334; 331].

In Chapter 7 to 9, I introduce three distinct MCC systems that I have developed. Chapter 7 is about a heterogeneous computing system that consists of a cluster of blade servers and a cluster of set-top boxes. It adopts a reliable and large-scale computing framework for data processing, called *Hadoop*. Using this framework, this new computing system can run very large data processing applications and data mining applications. In Chapter 8, I show a local training application for information extraction. It uses *Neural Network* to train for a *Natural Language Processing* algorithm. This algorithm runs on an MCC system where the parameters are first trained in the cloud and then further trained in a mobile device. In Chapter 9, I present an MCC application for audio stream retrieval. In this system, the database for retrieval is cached in the mobile device to reduce the queries sent to the cloud.

Chapter 3

Virtual Platforms

System simulation uses a software program that models the behavior of the target system. System simulation plays an important role in designing new systems. This has been proven in many different areas in system designs. Specifically, the design of embedded systems can benefit largely from using a simulation framework. There are many simulation frameworks that support the design of embedded systems. Simulation is particularly important in the development of embedded hardware and software. In embedded hardware development, simulation plays an important role as the method to pre-evaluate the prototyped design. Without simulation, the hardware designers have to make a real prototype device whenever they want to make changes to their design and test. In embedded software development, hardware simulation allows developers to test their software even before they have the finalized target hardware device. This allows the concurrent progress of hardware and software development, which saves a significant amount of time from the overall development process [29].

A virtual platform is a software model to simulate a target hardware system, in most cases an embedded system with a different microarchitecture from the host machine. Thus, it does not physically exist but behaves as a physical target system. Virtual platforms consist of simulated hardware components as well as internal modules to control and analyze the simulation, as shown in Figure 3.1. This is a full-system simulation where the entire system, or an embedded device, is modeled as a virtual platform. This simulation environment can actually execute the target software including embedded applications and operating systems, compiled for the target hardware architecture.

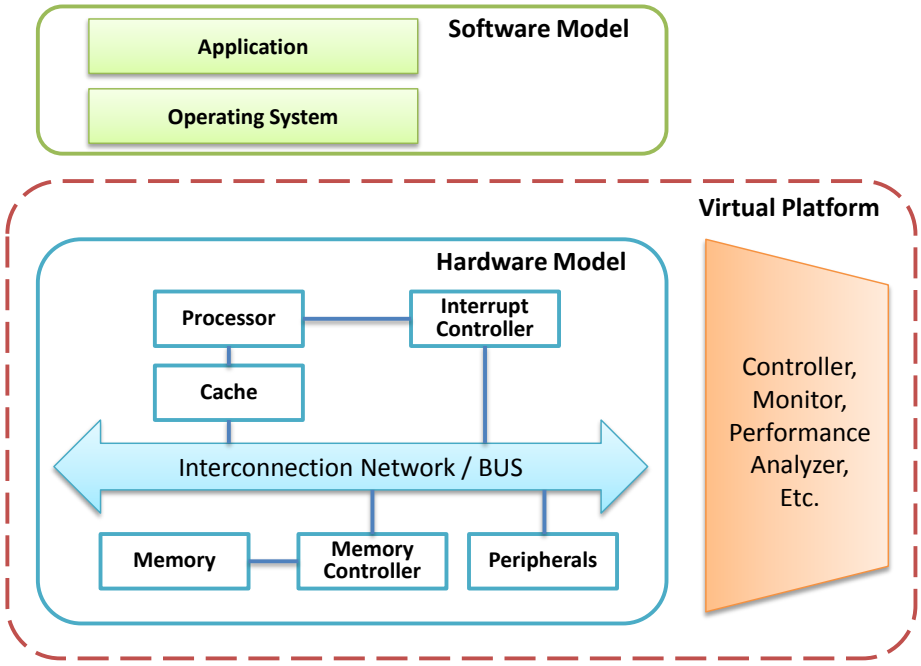


Figure 3.1: The architecture of a virtual platform.

In addition to simulating the embedded systems for their designs and application development, an increasing number of distributed systems is designed on simulation frameworks. Particularly, simulation frameworks that consist of multiple instances of virtual platforms are used to simulate multiple networked embedded systems, or distributed embedded systems. Thus, the importance of virtual platforms as a component in the design toolkit is also rising.

The scope of simulation that virtual platforms can provide varies from a single circuit, e.g., a computing processor, to a large, complex, system, e.g., a global-scale distributed system, as shown in Figure 3.2. In this chapter, I first introduce the use of virtual platform as a design tool for a single embedded system and then move on to a more complex usecase of virtual platform, the design of distributed systems using multiple virtual platform instances.

3.1 Advantages of Virtual Platform

There are many advantages of using a virtual platform (VP) for developing a new system:

- A VP enables early-stage application-software development for the target system without the need of having physical devices at hand. During this early period, the target hardware system

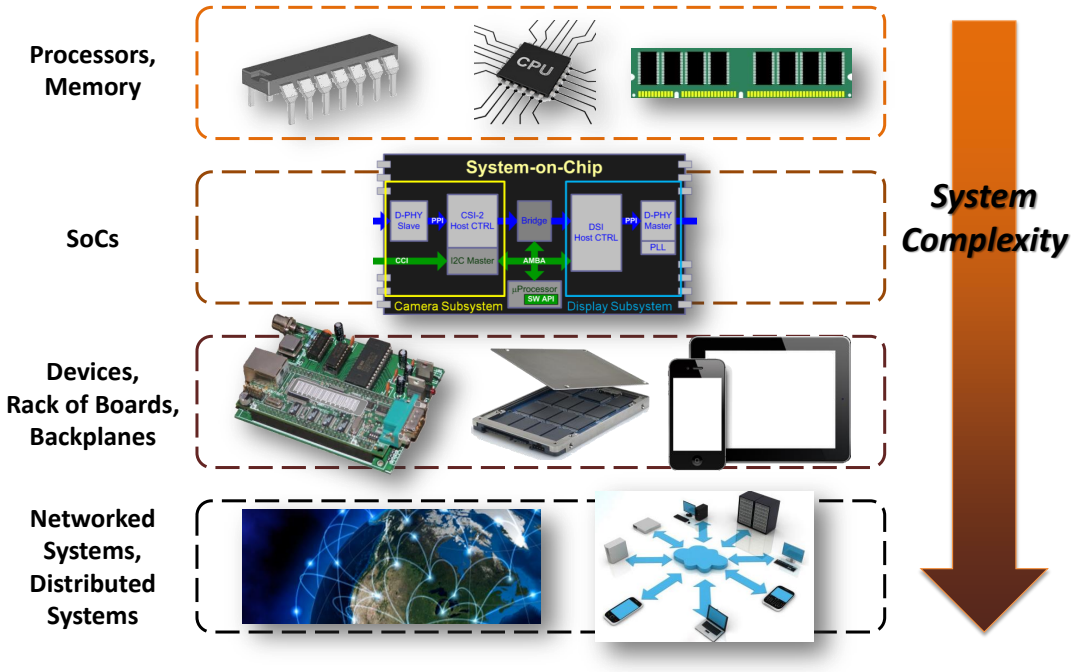


Figure 3.2: The different levels of complexity in simulation target systems.

is unlikely to be ready-to-ship. Thus, the use of a VP allows the parallel development of the target software and the hardware, therefore leading to shorter time-to-market.

- A VP reduces some overheads in the embedded software development process, such as cross-compilation, binary loading from the host machine to the target embedded system, or physical system configurations. This is because the VP is a piece of software and thus easier to be integrated with the automated development environment than physical hardware.

- A VP allows *Design Space Exploration* (DSE) of the target system. Therefore, while simulating the target software on the VP model, the system designer can flexibly adjust and decide the details of the target hardware modules to meet various constraints such as hardware costs, performance requirements, or energy consumption budget.

These advantages of virtual platforms play a large role in the system design, making it easy for the users to design a new hardware prototype, develop application software, and test the product.

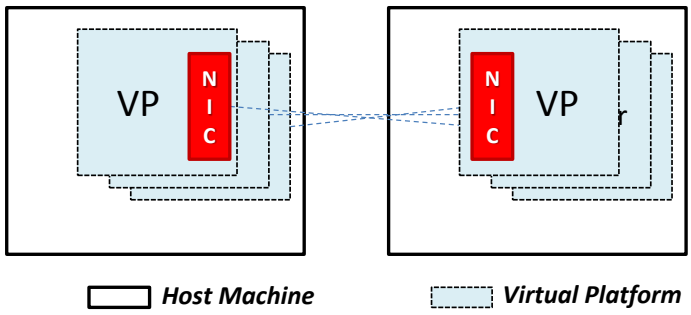


Figure 3.3: Multiple virtual platform instances connected through a network interface card.

3.2 Using Virtual Platform for System Development

Virtual platforms have been widely used in the design, development, optimization, and test of target hardware and software. Fast prototyping of the target system before designing detailed hardware is one of the most essential features of virtual platforms. Plug&Chip is a virtual framework for rapid prototyping of 3D SoCs [86]. Real-time systems also can be prototyped by using virtual platforms [230]. Kogel and Braun present a way to virtually prototype embedded systems for wireless and multimedia [186].

In many cases, a virtual platform is also used to optimize the hardware design of target systems by maximizing their performance through Design Space Exploration (DSE). For instance, Zuolo et al. propose SSDExplorer, a virtual platform-based simulator for DSE of Solid State Drives (SSDs) [357]. Meanwhile, another virtual platform is used to optimize the design of the memory controller in a 3D stacked wide I/O DRAM [163]. Hsu et al. present a virtual platform to optimally design a dual DSP core [143]. The objective of the optimization can vary. Particularly, a virtual platform can be used to optimize the power, thermal, and reliability of the target system [46].

There exist virtual platforms for other portions of the system development process. Ceng et al. offer a virtual platform-based solution for early-stage software development [61]. Some projects aim to support the co-development of the system platform and the software [338]. Meanwhile, testing is one of the important aspects of the development process. Virtual platforms can be used to stress-test the designed system [352; 91].

3.3 Networked Virtual Platforms for Distributed System Design

The previous section explained how using virtual platforms for developing a single, standalone system is a well-established, popular method. Recently, the use of virtual platforms is increasingly extended toward the development of systems where multiple devices interact through a communication network. This becomes possible thanks to the multiple virtual platforms that concurrently operate using their own model of network interface as shown in Figure 3.3. We name this collection of multiple virtual platform instances connected to each other *Networked Virtual Platform*. As each virtual platform can be extended to have peripherals, it features a network interface card module that is written as a piece of software and works as a physical network interface module.

There are multiple emerging projects that can be categorized as networked virtual platforms. Bakshi et al. presented MILAN, a model based extensible framework that facilitates the evaluation of a large class of embedded systems, through the seamless integration of different simulators [42]. Frølund and Garg presented Consul, a design-time simulation for a large-scale distributed object system [107]. They proposed a scalability analysis method to simulate distributed object systems. One of the major challenge in simulation-based designs is that the simulation results are not accurate because they rely on simulation parameters instead of measured values. They addressed this challenge by using the proposed analysis method that leverages relative and comparative reasoning to analyze design alternatives and compares transaction behaviors. Very recently, Sayyah et al. presented a virtual platform-based DSE for distributed embedded applications using multiple virtual platform instances [276]. Simics is one of the most popular solution that offers multiple virtual platform instances for the simulation of distributed systems [263].

In Chapter 4, I introduce a networked virtual platform, NETSHIP that I developed to support distinct levels of heterogeneity and large scalability. NETSHIP is the first kind of networked virtual platforms that supports the design, development, optimization, and test of large-scale distributed embedded systems. Particularly, NETSHIP can be used for MCC systems as it supports the modeling of virtual embedded systems as well as cloud computing environments through its VP-on-VM model [167].

Part I

Design and Simulation Tools for MCC Systems

Chapter 4

A Design Tool for Heterogeneous Large-Scale MCC Systems

From a single SoC to a network of embedded devices communicating with a backend cloud-computing server, emerging classes of embedded systems feature an increasing number of heterogeneous components that operate concurrently in a distributed environment. As the scale and complexity of these systems continues to grow, there is a critical need for scalable and efficient simulators. Together with Jihyung Park, Michele Petracca, and Luca Carloni, I developed a *Networked Virtual Platform* as a scalable environment for modeling and simulation. The goal is to support the development and optimization of embedded computing applications by handling heterogeneity at the chip, node, and network level. To illustrate the properties of our approach, I presented two very different case studies: the design of an Open MPI scheduler for a heterogeneous distributed embedded system and the development of an application for crowd estimation through the analysis of pictures uploaded from mobile phones.

4.1 Introduction

Computing systems are becoming increasingly more concurrent, heterogeneous, and interconnected. This trend happens at all scales: from multi-core systems-on-chip (SoC), which host a variety of processor core and specialized accelerators, to large-scale data-center systems, which feature racks of blades with general purpose processors, graphics-processor units (GPUs) and even accelerator

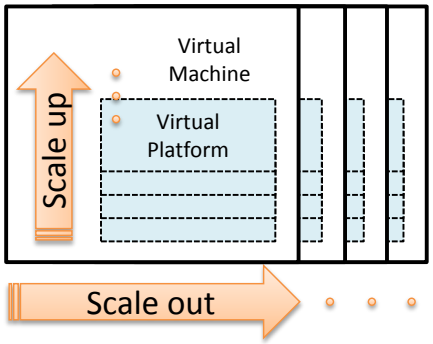


Figure 4.1: The two orthogonal scalabilities of NETSHIP.

boards based on FPGA technology. Furthermore, nowadays many embedded devices operate while being connected to one or more networks: e.g., modern video-game consoles rely on the Ethernet protocol [302], millions of TVs and set-top boxes are connected through DOCSIS networks [142], and most smartphones can access a variety of networks including 3G, 4G, LTE, and WLAN [190; 156; 320].

As a consequence, a growing number of software applications involve computations that run concurrently on embedded devices and backend servers, which communicate through heterogeneous wireless and/or wired networks. For example, *mobile visual search* is a class of applications which leverages both the powerful computation capabilities of smart phones as well as their access to broadband wireless networks to connect to cloud-computing systems [120; 314].

We argue that the design and programming of these systems offer many new unique opportunities for the electronic design automation (EDA) community. For instance, system and sub-system architects need tools to model, simulate, and optimize the interaction of many heterogeneous devices; hardware designers need tools to characterize the applications, software and network stack that they must support; and software developers need early high-level modeling environments of the underlying hardware architecture, often much before all its components are finalized.

As a step in this direction, we present NETSHIP, a networked virtual platform to develop simulatable models of large-scale heterogeneous systems and support the programming of embedded applications running on them. Users of NETSHIP can model their target systems by combining multiple different virtual platforms with the help of an infrastructure that facilities their interconnection, synchronization, and management across different virtual machines.

Given a target system, NETSHIP can be used to set up a simulation environment where each VP

works as single-device simulator running a real software stack, e.g., the Linux operating system, with drivers and applications. Thus, it makes it possible to run real applications over the entire distributed system, without actually deploying the devices. This allows users both to jump start the functional verification process of the software and to drive the design optimization process of the hardware and the network.

While in certain areas the terms *virtual platform (VP)* and *virtual machine (VM)* are often used without a clear distinction, in our research it is particularly important to distinguish them. A VP is a simulatable model of a system that includes processors and peripherals and uses binary translation to simulate the target binary code on top of a host instruction-set architecture (ISA). VPs enable system-level co-simulation of the hardware and the software parts of a given system before the actual hardware implementation is finalized. Instead, a VM is the management and provisioning of physical resources in order to create a virtualized environment. The resources are mostly provided by one or more server computers and the management is performed by a *hypervisor.* Examples of VPs include OVP, VSP, and QEMU, while KVM, VMware, and the instances enabled by the Xen hypervisor are examples of VMs. [1]

Thanks to its novel *VP-on-VM model*, the NETSHIP infrastructure simplifies the difficult process of modeling a system with multiple different VPs. In fact, the ability to support multiple VPs interconnected through a network makes NETSHIP free from the limitation of one specific VP while providing access to the superset of their features. For example, users who are interested in modeling an application running in part on certain ARM-based mobile phones and in part on MIPS-based servers can use NETSHIP to build a network of Android emulators [12] and OVP nodes.

The VP-on-VM model makes NETSHIP scalable both horizontally and vertically, as illustrated in Figure 4.1. The users can scale the system out by adding more VM instances to the network (horizontal scalability) and scale the system up by assigning to each VM instance more CPU cores on which more VP instances can run (vertical scalability).

Another pivotal advantage the VP-on-VM model adds to NETSHIP is access to the features of VMs, i.e., pausing, resuming the VM instances, duplicating instanced preconfigured for specific VP types, or migrating them across physical machines.

[1] Recent efforts to run VMs on embedded cores [80; 199] remain within the VM definition as they do not adopt binary translation.

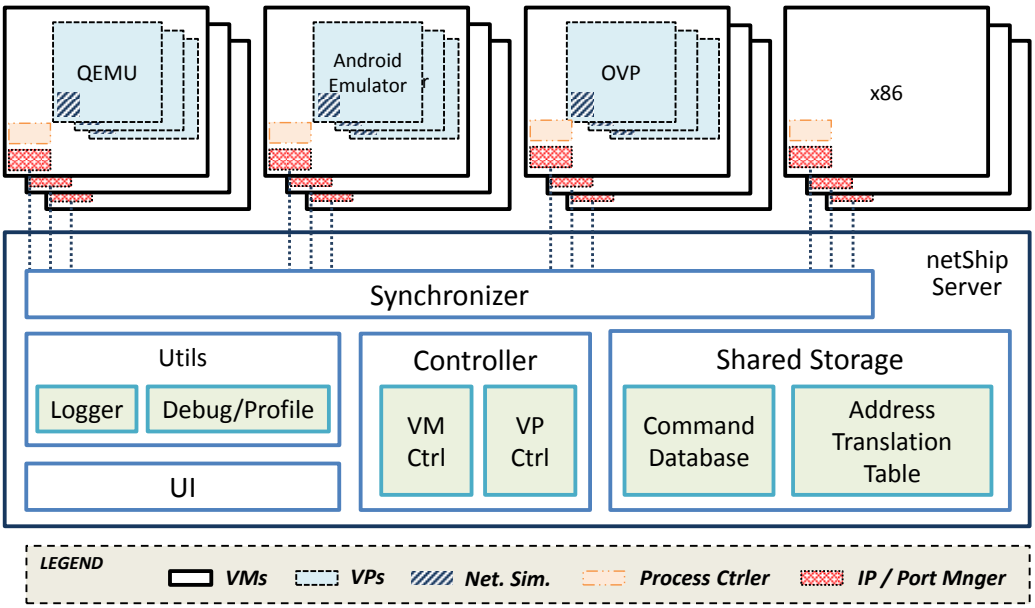


Figure 4.2: The architecture of NETSHIP.

Contributions. The main goal of this chapter is to understand how to build and use a *Networked Virtual Platform* for the analysis of distributed heterogeneous embedded systems. To do so, we built NETSHIP as a prototype based on the VP-over-VM model with the main objectives of supporting heterogeneity and scalability. This is the first work that presents this type of CAD tool. To evaluate NETSHIP we have completed a series of experiments including two complete case studies. The first case study shows how a networked virtual platform can be used to better utilize the computational resources that are available in the target system while guaranteeing certain performance metrics. The second case study shows how a networked virtual platform can be used to develop a software application running on a heterogeneous distributed system that consists of many personal mobile devices and multiple computer servers while, at the same time, obtaining an estimation of the resource utilization of the entire system.

4.2 Networked Virtual Platforms

A heterogeneous distributed embedded system can consists of a network connecting a variety of different components. In our approach, we consider three main types of heterogeneity: first, we are interested in modeling systems that combine computing nodes based on different types of processor

cores supporting different ISAs (*core-level heterogeneity*); second, nodes that are based on the same processor core may differ for the configuration of hardware accelerators, specialized coprocessors like GPUs, and other peripherals (*node-level heterogeneity*); third, the network itself can be heterogeneous, e.g., some nodes may communicate via a particular wireless standard, like GSM or Wi-Fi, while others may communicate through Ethernet (*network-level heterogeneity*.)

NETSHIP provides the infrastructure to connect multiple VPs in order to create a networked VP that can be used to model one particular system architecture having one or more of these heterogeneity levels. For example, Figure 4.2 shows one particular instance of NETSHIP which is obtained by connecting multiple instances of the QEMU machine emulator [51], the Android mobile-device emulator [12], and the Open Virtual Platform (OVP) [19].

Each VP instance runs an operating system, e.g., Linux, with all the required device drivers for the available peripherals and accelerators. The application software is executed on top of the operating system. Each VP typically supports the modeling of a different subset of peripherals: e.g., OVP supports various methods to model the hardware accelerators of an SoC: users can write models in SystemC TLM 2.0 or take advantage of the BHM (Behavioral Hardware Modeling) and PPM (Peripheral Programming Model), which are C-compatible Application Programming Interfaces (APIs) that can be compiled using the OVP-supplied PSE tool-chain[2].

In addition to the features supported by each particular VP, we equipped NETSHIP with all the necessary instrumentation to: (1) enable multiple instance executions; (2) configure port forwarding; and (3) measure the internal simulation time. Furthermore, any node in the network of VPs could potentially be a real platform, instead of being a virtual one: e.g., in Figure 4.2, each of the x86 processors runs native binary code and still behaves as a node of the network.

One of the main novelty aspects of NETSHIP is the *VP-on-VM model* which is critical for the scalability of modeling and simulations. We designed NETSHIP so that multiple VP instances (e.g., 2 to 8) can be hosted by the same VM. By adding more VMs, the number of VPs in the system can be increased with a small performance penalty, as discussed in Section 4.3. Notice that the simple action of cloning a VM image that includes several VPs often represents a convenient way to scale out the model of the target system.

Next, we describe the main building blocks of NETSHIP.

[2]PSE is Imperas Peripheral Simulation Engine [19].

Synchronizer. VPs vary in the degree of accuracy of the timing models for the CPU performance that they support. Some VPs do not have any timing model and simply execute the binary code as fast as possible. This is often desirable, particularly when a VP runs in isolation. In NETSHIP, however, we are running multiple VPs on the same VM and, therefore, we must prevent a VP from taking too much CPU resources and starving other VPs. QEMU provides a crude way to keep simulation time within a few seconds of realtime. OVP, instead, controls the execution speed so that the simulated time never surpasses the wall clock time. Multiple OVP instances, however, still show different time developments which require a synchronization method across the VPs in the network.

We equipped NETSHIP with a *synchronizer* module to support synchronization across the heterogeneous set of VPs in the networked platform, as shown in Figure 4.2. The synchronizer is a single process that runs on just one particular VM and is designed in a way similar to the fixed-time step synchronization method presented by D'Angelo et al. [82]: at each iteration, a central node increases the base timestamp and the client nodes stop after reaching the given timestamp. However, we considered two aspects in our synchronizer:

- we must synchronize VPs that might be scattered over several physically separated machines;
- we must preserve the scalability provided by the VP-on-VM model.

NETSHIP targets large-scale systems which involve deployments across physically separated machines where millisecond-level network packet travelling is actually required to synchronize. Hence, NETSHIP supports the modeling of applications that have running times ranging from a few seconds to multiple hours or days, rather than simulations at nanosecond-level.

To support synchronization over the VP-on-VM model, we designed a *Process Controller* (PC) that allows us to manage the VPs in a hierarchical manner. Each VM hosts one PC, which controls all the VPs on that VM. In particular, all messages sent by a VP to the synchronizer pass through the PC. The PC supports also running programs on a host machine: e.g., in the case of Figure 4.2, the PCs manage the synchronization of the processes running on a x86 through the two POSIX signals `SIGSTOP` and `SIGCONT`, in the same way as the UNIX command `cpulimit` limits the CPU usage of a process.

Figure 4.3 illustrates an example of the synchronization process with two VMs, each hosting two VP instances. The following steps happen at each given iteration i:

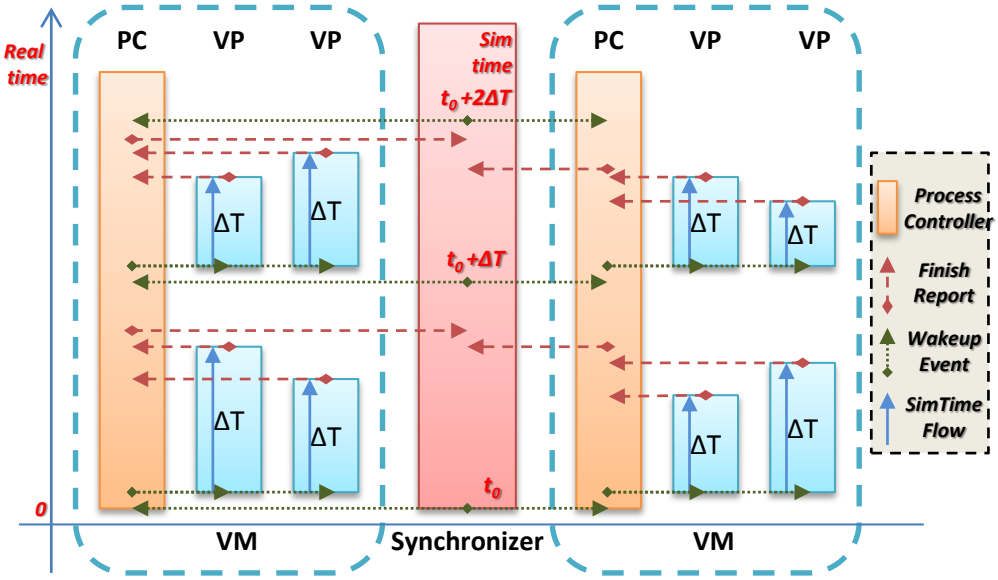


Figure 4.3: Synchronization process example.

1. the synchronizer issues a future simulation time $t_i = t_{i-1} + \Delta T$ to the VPs and wakes them up;
2. the VPs run until they reach the appointed time t_i and report to their PC;
3. as soon as a PC receives reports from all the connected VPs, it reports to the synchronizer;
4. after the synchronizer has received the reports from all the PCs, it loops back to Step 1.

The users can configure the time step ΔT to adjust the trade-off between the accuracy and the simulation speed.

Let's compare the complexity of this hierarchical method to the existing method. In the previously proposed synchronization algorithm [82], if the number of VP is $|VP|$, then the synchronization process should receive and count $|VP|$ reports to make sure that all the VPs have reached to the appointed simulation time. This results in a $\Theta(|VP|)$ algorithm complexity in *Synchronizer*. In contrast, the complexity of NETSHIP is $\Theta(\sqrt{|VP|})$ because *Synchronizer* manages $\sqrt{|VP|}$ PCs, each of which controls $\sqrt{|VP|}$ VPs.[3]

[3]It may be enough for *Synchronizer* to count only the number of reports from PC to know that every VP instance is ready and advance the simulation time. However, this method is unreliable in the sense that there is no way for *Synchronizer* to tolerate a PC malfunctioning. If a hash table, for example, is used to map a PC's IP to the data structure for checking that the PC is reporting more than once in a cycle, the average complexity of the algorithm in [82] is $O(|VP| + |VP|^2/k)$ while for our algorithm it is $O(\sqrt{|VP|} + \frac{|VP|}{k})$, where k is the number of buckets in the hash

Command Database. NETSHIP was designed to support the modeling of systems with a large scale of target networked VPs. In these cases, to manually manage many VP instances becomes a demanding effort that involves many tasks, including: add/remove new VP instances to/from a system, start the execution of applications in every instance, and modify the configuration files in the local storage of each instance. In order to simplify the management of the networked VP as a whole, we developed the *Command Database* that stores the script programs used by the different NETSHIP modules. For example, the network simulation module and IP/Port forwarding module load the corresponding scripts from the database and execute them. Table 4.1 contains a detailed list of the commands in the database.

Name	Behavior
vp_ctrl_pwr	turns the VP on/off
net_set_bw	sets the VP's network bandwidth to simulate
net_set_delay	sets the VP's network delay to simulate
net_set_error	sets the VP's network error rate to simulate
net_load_rt	loads the address/port settings to use
cmd_execute	executes a command in all the VPs
acc_gen	loads driver modules and creates a device node for the specified accelerator
report_local	reports the local time in the VP
report_cpu	reports the cpu time in the VP

Table 4.1: List of commands in the command database.

VM and VP Management. Whereas the commands in the Command Database are dedicated to VP configuration, we developed specialized modules to manage the VPs and the VMs (for the latter we integrated tools provided by the VM vendor). These modules manage the disk images of the VMs and VPs, for creating, copying, and deleting their instances. Since many VPs are still in the early stages of development and are frequently updated by the vendors, the VP management module checks the availability of new updates for all the installed VPs.

Network Simulation. The VP models of NETSHIP are provided with their own models of the network interface card (NIC). These models, however, are purely behavioral and do not capture any

table and searching n times in a hash table takes $n * O(1 + n/k)$.

network performance property, such as bandwidth or latency [82]. Consequently, we developed a *Network Simulation* module that enables the specification of bandwidth, latency, and error rates, thus supporting the modeling of network-level heterogeneity in any system modeled with NETSHIP. As shown in Figure 4.2, a Network Simulation module resides in each particular VP and uses the traffic-shaping features based on the *tc* command, which manipulates the traffic control settings of the Linux kernel.

Library	Option	Default Value
SSH (fixed)	Port	22
Hadoop	dfs.http.address	50070
(fixed)	dfs.datanode.http.address	50075
	mapred.job.tracker.http.address	50030
	mapred.task.tracker.http.addres	50060
Open MPI	oob_tcp_port_min_v4	0
(random)	oob_tcp_port_range_v4	65535
	btl_tcp_port_min_v4	1025
	btl_tcp_port_range_v4	65525

Table 4.2: Example of library port uses.

Address Translation Table. In NETSHIP there are two points where packet forwarding plays a critical role:

1. To allow incoming connections to the VPs through their emulated NIC model, most VPs provide a way to redirect a port of the host to a port of the VP, so that packets that arrive to that VM port are redirected to the corresponding VP port. We leverage this redirection mechanism so that the applications running on the VPs can open ports to receive packets from other VPs, even if those are located on a physically separated VM. [4]

   Port forwarding is the technique of redirecting the traffic incoming on one network port of the OS running on the host VM towards a specific port of the OS running on the hosted VP. For example, when a packet arrives to Port 10020 of the VM's OS, the VP to which Port 10020 is assigned intercepts the packet and forwards it to Port 22 of the VP's OS. Hence, when users

[4]While certain VPs provides a network bridge feature that allows more generic network functionalities, we use port redirection because it is commonly supported by every VP family.

connects through SSH to the host's IP and Port 10020 they are forwarded to Port 22 of the VP. This is configured in the behavioral model of the VP and performed through the NIC model.

Unlike SSH, some libraries require a random port to be accessed by clients; for instance Open MPI communicates through random ports ranging from 1025 to 65535 [124]. However, most libraries also provide a way to change or reduce the required port range as shown in Table 4.2. We reduced the range and mapped it to the same port range on the virtual addresses, 200.0.0.x. One of these addresses is allocated to each of the VP instances using *iptables* through the Port Management module in Figure 4.2.

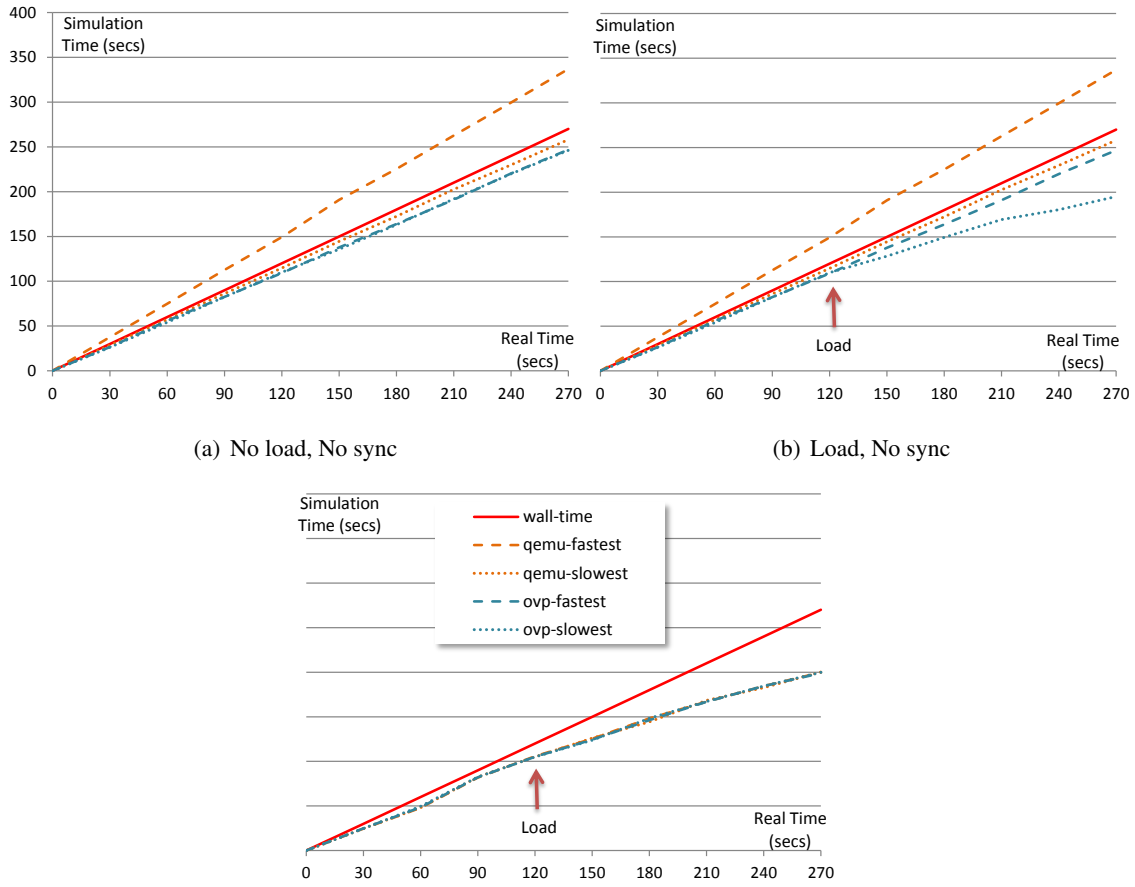
2. Since certain applications require that each VP must be accessible through a unique IP address and generally there is only one physical IP address per VM, we must map each VP to a virtual IP address. Each VP must know such mapping for all other VPs in the system. Hence, we used the UNIX command `iptables` to create a table of assignments within the kernel of each VP. NETSHIP stores the translation information in the *Address Translation Table*, which is loaded through the network commands stored in the Command Database.

4.3 Scalability Evaluation

In this section, we demonstrate the capabilities of NETSHIP from the synchronization, scalability, performance, and network-fairness perspectives. Functional validations of NETSHIP will be covered in each of the two case studies, in Section 4.5 and 4.6, respectively.

Simulated Time and Synchronization. Eight OVP instances and eight QEMU instances are running in this simulation setup. The three figures in Figure 4.4 show the *simulated time* for each instance. The red solid line represents the time graph of an ideal VP, with $y = x$, where y is the *wall-clock time* and x is the simulated time. While there are multiple instances running together, in the figure we show only the fastest and slowest instances for each VP family, in order to summarize the range of variations within each VP family and to better compare the VP families.

Figure 4.4(a) measures the simulated time of the unloaded VPs. Each VP advances its simulated time linearly, but differently from each other. In particular, the range of simulated time among QEMU instances is wide: from 4% slower up to 25% faster than the wall-clock time. Instead, the OVP instances show almost the same simulation speed (0.3% variation), which is 8% slower than



(a) No load, No sync

(b) Load, No sync

(c) Load, Sync

Figure 4.4: Simulation time measurements.

the slowest QEMU instance. This reflects the fact that OVP has a better method to control the simulation speed.

Figure 4.4(b) shows the case when a VP is subject to a heavy workload. In particular, at simulated time $x = 120s$ one OVP instance starts using a high-performance accelerator. From that point on, that OVP instance gets slower than every other instance, as shown by the deviation among the OVP lines in the figure. This is natural when the peripherals are modeled at a very high level of abstraction. In a fair host VM, all VPs are granted the same amount of CPU time to be executed. Simulating the use of a hardware accelerator on a VP typically requires the VP process on the VM to execute a non-negligible computation. In the other words, running the functional model of the accelerator uses the VM's CPU resources and requires a certain amount of wall-clock time. From the viewpoint of simulated time, however, this computation happens in a short period of time (due to the

VP Type	Core Model	CPU use	Preferred #VPs
OVP	Accelerator	~ 24%	4
OVP	MIPS	~ 6%	16
QEMU	PowerPC	~ 12%	8
VMWare	x86	~ 5%	20

Table 4.3: Host CPU use of each VP.

accelerator's timing model); therefore the given VP instance becomes slower than the others. The misalignment of the simulated time among VP instances is a concern when simulating distributed systems, because it might cause the simulated behaviors to be not representative of reality.

To address this problem, we implemented the synchronization mechanism explained in Section 4.2. Figure 4.4(c) shows the behavior of all VP instances under the same conditions but with the synchronization mechanism turned on (with a synchronization cycle of 300ms). The simulated time of all VPs becomes the same as the slowest instance. The synchronization cycle can be decided by the users. Our experiments show that it should not be too small ($\geq$ 1ms) because: i) a synchronization that is much more frequent than the OS scheduling time slice[5] may disturb the timely execution of the VPs, and ii) the synchronization is an overhead and slows down the overall simulation.

Vertical Scalability. By *vertical scalability* we mean the behavior of the networked VP as more VPs are added to a single VM. As discussed above, although the synchronizer preserves the simultaneity of the simulation among VPs, it makes them all run at the speed of the slowest instance , i.e., even one slow VP instance is enough to degrade the simulation performance of the whole system. Therefore, an excessive number of VP instances on the same VM will likely cause a simulation slowdown.

Table 4.3 shows the amount of CPU of the host VM that is used by a VP instance. For example, when an OVP instance fully utilizes one accelerator, it takes up to 24% of the host CPU resource in the hosting VM. This means that four is the optimal number of OVP instances, equipped with that accelerator, which can co-exist on the same VM without performance penalty. Likewise, the

[5]Linux O(1) scheduler dynamically determines the time slice, ranging from milliseconds to a few hundreds milliseconds.

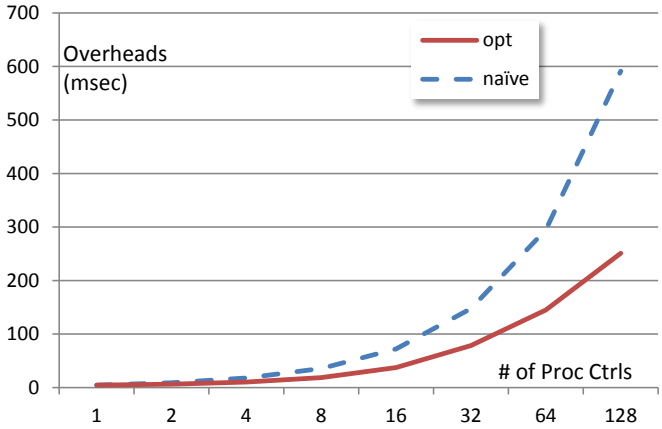


Figure 4.5: Synchronization overheads.

CPU of a QEMU PowerPC that is fully busy, i.e., a simulated 100% utilization, uses up to 12% of the host VM's CPU resources: hosting up to eight QEMU PowerPC instances in the same VM is performance optimal.

Note that even if the number of VP instances is higher than the optimal value, the synchronizer still preserves the simultaneous simulation of all nodes. However, balancing out the number of VP instances hosted across the VPs, or alternatively increasing the computational resources available to the VM, helps to increase the overall simulation performance.[6]

Synchronization Overheads and Horizontal Scalability. *Horizontal scalability* describes the behavior of the simulated VP as we scale the number of VMs. The synchronizer is the entity in the networked VP that communicates with all VMs in order to keep all VPs aligned. Figure 4.5 shows the overhead increase as the number of VMs grows. We measured the overhead as the time elapsed from when the slowest VP instance reports the termination of its the execution step to the time the same instance starts a new one. We experimented with ten VP instances insisting on each PC. In the figure we compare a naïve implementation with an optimized implementation of the synchronizer. For both versions the principle of the synchronization is the same; however, in the optimized version we used more advanced techniques to reduce the communication latency and overhead, using the following methods:

Multicasting-based Wakeup. In order to reduce the serial latency of the wakeup packets de-

[6]The CPU resources of the VM might not be the only bottleneck. For a more generic approach, an analysis of disks, network congestion, memory bandwidth, bus capacity, and cache interference is required. In our experiments, however, the constraint due to the VM's processing power was the most dominating factor that decides Vertical Scalability.

livered from the synchronizer to the PCs, we used multicast UDP.

Atomic Operations in Shared Memory for In-Machine Reporting. The PC must check that VPs correctly report the end of the current simulation cycle. This is done by having each VP increase a shared counter through an atomic operation. This is possible because all VPs are on the same machine.

Disabling Nagle's Algorithm. Unlike the waking-up message of the synchronizer to the PCs (1-to-N), multicast UDP cannot be used to carry reports from the PCs to the synchronizer (N-to-1). In the Linux kernel, TCP sockets typically use by default an optimization technique, Nagle's algorithm, which combines a number of small outgoing packets and sends them all in one single message [232]. This method, however, increases the latencies of these small packets (up to 30ms in our experiments), which is a critical issue in our synchronization design, since latency is more important than throughput. We then disabled Nagle's algorithm by turning on the socket option TCP_NODELAY for each TCP socket.

Using POSIX Signals to Sleep and Wake up. In order to stop and wake-up a VP instance our PC uses two signals: *SIGSTOP* and *SIGCONT*. The use of standard Linux signals provides several advantages. First, the PC can be easily implemented in a separate user-space program, without the knowledge of the internals of the VPs. Second, once implemented, the PC is portable across the VPs, requiring no modifications. Third, the PC can stop all threads in the process, while sleeping works only for the thread of the current context. Most importantly, this also enables a synchronized execution with processes that run natively on a host VM, e.g., x86 server, outside of any VP.[7]

The overhead for 128 PCs is approximately $250ms$, which slows down the networked VP by about 25% if the simulation step is set to $1s$.

Although the optimized implementation significantly reduces the overhead, both slopes increase linearly with respect to the number of PCs in the network (notice that the x-axis is logarithmic). This is because synchronization involves all PCs, each of which is located in a separate machine, and all reports require a packet transmission across the network and linear-time computation to parse the reports.

In summary, the synchronization across VMs limits the horizontal scalability, in the sense that

[7] In NETSHIP x86 binaries are executed on a VM, not a VP. Through the stop and continue signals PC synchronizes the process without modifying the binary executables.

Target	PNI[8]	RTT (ms)
self VP (the loopback interface)	no	0.15
local VM (a VM where the testing VP runs)	no	0.17
local VP (another VP on the local VM)	no	0.19
remote VM (a VM, except the local VM)	yes	0.17
remote VP (a VP on the remote VM)	yes	0.19

Table 4.4: Ping test from a VP.

the simulation step after which all VPs are synchronized must be (much) bigger than the time it takes to actually perform the synchronization, which strongly depends on the characteristics of the hosting VMs and how they are connected.

4.4 Experiments

4.4.1 Network Fairness Depending on Deployment

Based on the measurement of the latency of packet responses, through tools such as *ping* or *traceroute*, it is possible to determine whether two VM instances are deployed on the same physical machine or not [264]. Likewise, VP instances may experience variations in the network latency, depending on how they are deployed.

However, our experimental results in Table 4.4 show that, given a VP instance, the difference in latency to reach a *local VP* on the same VM, versus a *remote VP* on another VM, is less than $0.05ms$. In particular, this value is small enough to be effectively hidden by the latencies set by the designer to model the network-level heterogeneity configuration, as shown in Table 4.6.

4.4.2 Scalability and Detailed Configurations

In general, Horizontal Scalability is the ability to have more VP instances running in NETSHIP by adding more VM instances. Vertical Scalability is the ability to have more VP instances running in NETSHIP by adding more CPU cores to a VM. For example, the preferred number of OVP instances with accelerators that can run in one VM (with one CPU core) is four, as shown in Table 4.3. If we

[8]Physical Network Involved.

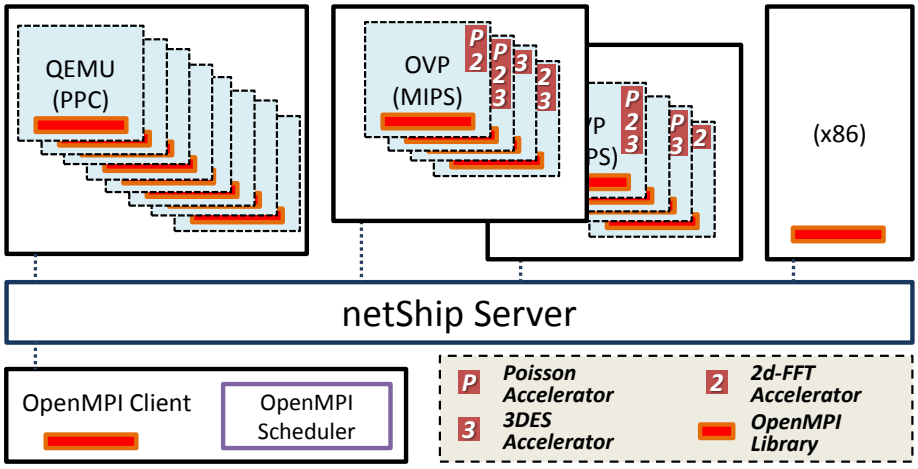


Figure 4.6: The system architecture for Case Study I.

add one more CPU core to the VM, we can run up to eight OVP instances with accelerators in that VM.

The configuration of Case Study I, shown in Figure 4.6, includes: one VM that runs eight QEMUs, two VMs that run four OVPs each, and one VM that supports x86. This is an optimal configuration for the purpose of this case study. However, we also test Horizontal Scalability (e.g., by adding one VM that runs eight other QEMUs and another VM that runs four other OVPs) and Vertical Scalability (e.g., by adding one CPU core to the VM running eight QEMUs so that it can sustain up to 16 QEMUs.)

For Case Study II we vary the number of Android Emulator in "Android Emulator Scalability" and the number of OVP instances in "Bottleneck Analysis" in Section 4.6. We run one to five Android Emulators on one VM. We use two CPU cores for each VM, hosting four OVPs on each CPU core. Thus, we use a total of four VMs for 32 OVP instances.

4.5 Case Study I - MPI Scheduler

We modeled a distributed embedded system as a networked VP that runs Open MPI (Message Passing Interface) applications. This system features all the three kinds of heterogeneities: it has three different models of CPUs, it has arbitrarily scattered accelerators over the VPs, and we also vary the types of network connecting the devices. The goal of this case study is to use a networked VP for designing a static scheduler that optimizes the execution time by better distributing the work

Algorithm	Operations per Hour		Speedup
	CPU	Accelerator	
Poisson	349	1183	3.39
2d-FFT	314	517	1.65
3DES	632	1339	2.12

Table 4.5: Case Study I: performance comparisons.

across the system in Figure 4.6.

Open MPI is an open source MPI-2 implementation, which is a standardized and portable message passing system for various parallel computers [18]. In this case study, we used Open MPI to establish a computation and communication model over NETSHIP. Since the mainstream implementation of Open MPI does not support MIPS or ARM architectures (because it misses the implementation of atomic operation backends) we wrote and applied patches for Open MPI to run on these ISAs.

We simultaneously run three MPI applications over the distributed system: Poisson [235], 2d-FFT [290], and Triple DES [44]. Each application is a standalone executable program and is configured to process a small amount of data so that they act as embarrassingly parallel. The binary code is stored in the NETSHIP Server's shared storage, from which each VP downloads a copy when the system starts-up. Every application is designed to either use the hardware accelerator, whenever available on the VP, or run purely in software, otherwise. Accelerators are modeled to run basically the same algorithm as the applications. We modeled one iteration of the algorithm to take a few milliseconds.

According to our timing model for the accelerators and to the native timing model of the CPUs, accelerators show $1.65\times \sim 3.39\times$ speedup over CPUs, as summarized in Table 4.5. Note that the speedup introduced by hardware acceleration with respect to pure software execution is not the main point here. Instead, the comparative analysis is a demonstration of the type of analysis that a designer can carry by using the networked VP paradigm. In fact, the speedup mentioned above is actually conservative with respect to the literature, in order to keep the design exploration of our case study interesting [288; 282]. The 2d-FFT accelerator's performance improvement is not as high as the other two accelerators because i) the size of the input and output of the 2d-FFT algorithm

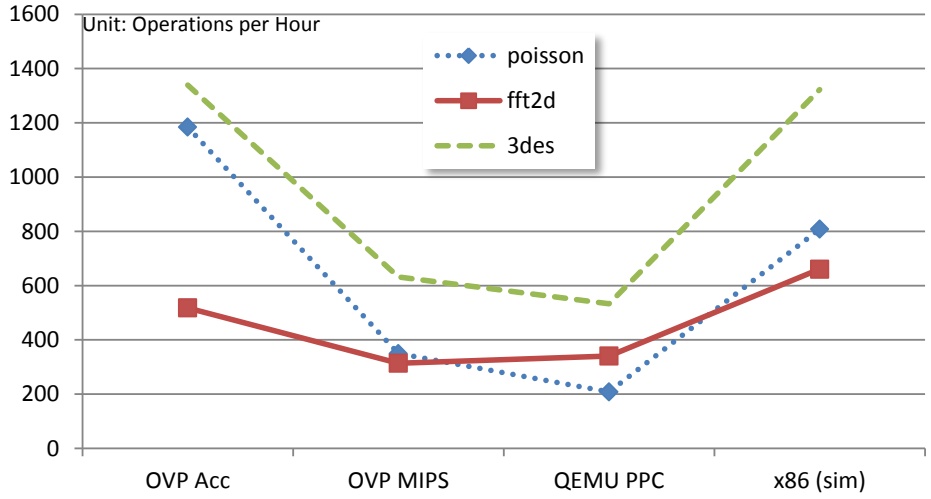


Figure 4.7: Case Study I: performance of different cores.

VP Type	Network Type	Bandwidth	Latency
OVP MIPS	DOCSIS 2.0	30.72Mbps	30ms
QEMU PowerPC	Evolved EDGE	1.00Mbps	80ms
Host x86	IEEE 802.11g	54.00Mbps	45ms

Table 4.6: Case Study I: configured bandwidth & latency.

is larger than other accelerators and ii) the execution time for this algorithm is longer, according to our timing models.

Based on such time model, Figure 4.7 shows the performance profile for different applications on a few VPs: the OVP instances are always equipped with an accelerator, while the other VPs are not. We also consider models for the network, whose bandwidth and latency parameters are summarized in Table 4.6. Note that, since NETSHIP allows designers to use time models to better simulate the characteristics of the network, those models are inputs to the networked VP and their derivation goes beyond the scope of this thesis.

In order to improve the application performance by taking advantage of the known properties of the system, i.e., performance profile of the nodes and network characteristics, we designed an *OpenMPI Scheduler* and we used the networked VP to evaluate its effectiveness. As shown in Table 4.7, the scheduler delivers a speedup ranging from $1.3\times$ to over $4\times$, depending on the user request. Such achievement is very encouraging since it is obtained without using any additional

# of operations			Time in ms		
Poi-sson	2d FFT	Triple DES	without Sched	with Sched	Speed Up
60	30	800	199,642	48,924	4.08
60	30	1800	344,422	117,380	2.93
50	20	40	102,293	45,210	2.26
60	30	150	103,700	46,693	2.22
250	80	140	198,927	113,383	1.75
20	100	10	161,462	122,527	1.32

Table 4.7: Case Study I: scheduler performance.

resource, but only by re-assigning tasks to the nodes that are better equipped for each of them. Note that the design, the verification, and also an initial assessment on the effectiveness of the scheduler have been carried out on the networked VP, without having to deploy the real system.

Scheduler Design. We designed the scheduler to run on a client machine and to follow these steps:

1. It receives the *user request*, which includes the number of times each MPI application must run, e.g., 300 Poissons, 200 2d-FFTs, and 500 3DESs.
2. It loads the *performance profile* of each VP in executing the given applications, e.g., VP_0 takes 3.182s to execute Poisson, while VP_1 takes 10.427s , and so on, as shown in Figure 4.7.
3. It derives the objective function to be minimized.
4. It converts the constraints (user request and performance profile) and the objective function into a matrix, and solves it by running a linear programming algorithm.
5. It distributes the workload according to the solution.

For the sake of simplicity, the following two examples assume that there are two devices, d_1 and d_2, and two applications, x_1 and x_2. In both examples, variables T_{ij} denotes the amount of time that device d_i spent on an application x_j. For instance, a variable T_{12} is the time period that device d_1

$T11$	T_{12}	T_{21}	T_{22}	t	
110	0	150	0	0	400
0	250	0	100	0	320
-1	-1	0	0	1	0
0	0	-1	-1	1	0
0	0	0	0	1	0

Table 4.8: Linear programming matrix for minimizing execution time.

spent running application x_2. After executing a linear programming algorithm, the solution includes the best (the minimun or the maximun) possible value of the objective function along with the values of the variable used for the best value of the objective function.

Minimizing the Execution Time. Let's assume we have the following profile data:

1. application x_1 runs on device d_1 110 times per unit time.
2. application x_2 runs on device d_1 250 times per unit time.
3. application x_1 runs on device d_2 150 times per unit time.
4. application x_2 runs on device d_2 100 times per unit time.

The user's request may be like the following:

1. Execute application x_1 400 times and application x_2 320 times.

Then we have two inequalities:

1. $110T_{11} + 150T_{21} >= 400$
2. $250T_{12} + 100T_{22} >= 320$

To account for the total execution time, we introduce a new variable t.

1. $T_{11} + T_{12} <= t$
2. $T_{21} + T_{22} <= t$

Finally, the objective function p to be minimized is equal to t. The resulted matrix is given in Table 4.8. The optimal solution for this example is $p = 52/25$; $T_{11} = 4/5$, $T_{22} = 32/25$, $T_{21} = 52/25$, $T_{22} = 0$, $t = 52/25$. This means that the devices can finish the user request in 52/25

T_{11}	T_{12}	T_{21}	T_{22}	
110	0	150	0	400
0	250	0	100	320
30	50	20	70	0

Table 4.9: Linear programming matrix for minimizing power dissipation.

time units when the system follows this solution. The solutions for each variable, as the definition, stand for the execution time, e.g., T_{11} requires device d_1 to run application x_1 for 4/5 unit time or 88 times.

Minimizing the Power Dissipation. In this example, we assume the same user request and profile data used in the execution time minimization example. In addition to these conditions, the power dissipation profile data is required:

1. device d_1 dissipates 30W per unit time when executing application x_1.
2. device d_1 dissipates 50W per unit time when executing application x_2.
3. device d_2 dissipates 20W per unit time when executing application x_1.
4. device d_2 dissipates 70W per unit time when executing application x_2.

Then, the objective function derived is $p = 30T_{11} + 50T_{12} + 20T_{21} + 70T_{22}$. The corresponding matrix to minimize this objective function subject to the constraints is given in the Table 4.9.

The execution of a linear programming algorithm for this matrix gives the solution $p = 352/3$; $T_{11} = 0$, $T_{12} = 32/25$, $T_{21} = 8/3$, $T_{22} = 0$. This means that the user requests can be executed with consuming $352/3$ power units.

4.6 Case Study II - Crowd Estimation

Crowd estimation, or crowd counting, is the problem of predicting how many people are passing by or are already in a given area [216]. A number of researches have focused on crowd estimation based on image processing of pictures [99; 203]. The crowd estimation application we developed in this section is based on user-taken pictures, from mobile phones, targeting relatively wide areas, e.g., a city.

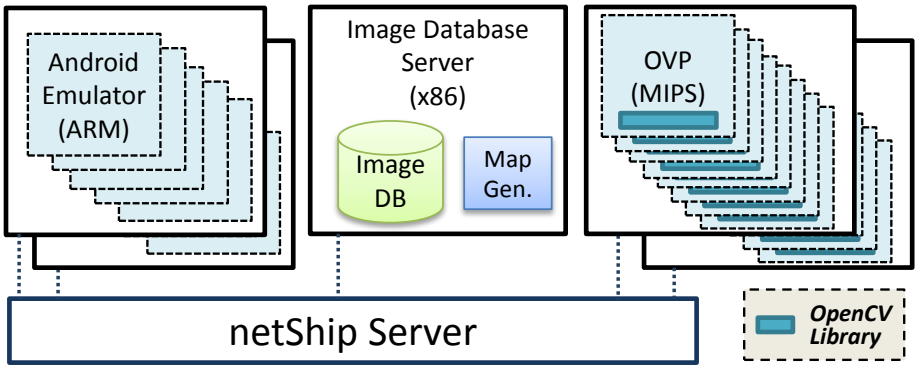


Figure 4.8: The system architecture for Case Study II.

We built a networked VP (Figure 4.8) that is representative of the typical distributed platform required to host this kind of application. The networked VP features *Android Emulator*s to model the phones and a cluster of *MIPS-based servers* based on the multiple OVP instances (on the right-hand side of the figure). The Android Emulators emulate mobile phones that take pictures through the integrated camera and upload them to the cloud. The pictures are stored on an *Image Database Server* (IDS), to which both phones and servers have access. The servers emulate the cloud, and run image processing algorithms on the pictures. Specifically, we developed a *Human Recognition* application based on OpenCV [20] to count the people in each picture and store the result on the IDS. Then, a *Map Generator* process running on the IDS reads the people counting from the IDS and plots it on a map.

Given the application requirements, we used the networked VP to gain insights on the amount of resources required to process pictures in real-time. Note that our main concern is the opportunity to build and study the networked VP and to use it to analyze the properties of the application that runs on it. In other words, we used this application primarily as a case study to test the capabilities of NETSHIP, while the optimization of the quality of the crowd estimation was only a secondary concern.

Android Emulator Scalability. We used several Android Emulators to model millions of mobile phones that sporadically take pictures (instead of using millions of emulators). To validate whether the emulators realistically reflect the actual devices' behavior with respect to network utilization, we performed multiple tests after making the following practical assumptions:

1. There are 3,000,000 mobile phone users in Manhattan and 2% of them upload 2 pictures a

# of Emulator in the model	# of Pic Upload / Hour	Incoming Traffic (KB/s) Max	Min	Avg.
1	13333	N/A	N/A	N/A
2	6666	380.4	372.5	379.3
4	3333	384.1	376.4	381.2
8	1666	377.8	361.8	374.2
16	833	389.0	367.5	381.5

Table 4.10: Case Study II: impact of varying number of Android-emulator instances.

Image Size (KB)	8	32	128	512	74(Avg)
Process Time (s)	3.48	13.42	49.7	247.1	31.5
Throughput (KB/s)	2.30	2.38	2.57	2.07	2.34

Table 4.11: Case Study II: image processing (human recognition) performance.

day.

2. The uploading of the pictures is evenly spread over the daytime (09:00∼ 18:00).
3. The average image file size is 74KB, as the image size we have in the database (DB).

Given the assumptions above, in Table 4.10 we summarize the number of pictures an emulator must upload in an hour and the actually measured incoming traffic of the DB server. These values are obtained based on the number of available emulators in the networked VP and accordingly configured the number of pictures uploaded by each emulator per hour. For example, if the networked VP has only one Android emulator (first row), we can achieve the desired load for the cluster when this emulator uploads $3,000,000 * 0.02 * 2/9 \approx 13333$ pictures per hour. Since one single emulator fails to upload 13333 pictures per hour, because of insufficient emulator performance, we must increase the number of emulators to at least two. The measured incoming traffic is rather consistent independently of how many emulators we use to split the job. This implies that we can deploy less emulators than the number of nodes we would have in reality, i.e., 4 vs. 3 million, as long as those emulators generate more traffic than they would in reality, i.e., 3333/hour vs. 2/day, after verifying that they also simulate fast enough to sustain the traffic generation.

Bottleneck Analysis. The data in Table 4.11 show the average time required by one MIPS server to run the Human Recognition application on a given picture. Based on this data and on

the characterization of the traffic load, the application designer can attain a number of meaningful design considerations.

1. The designer can measure the number of required MIPS servers that support the required volume of image processing, given the input and output data rates. For example, when images are fed to the DB at $380KB/s$, then, based on the throughput for the average image size in Table 4.11, the cluster must have more than $380KB/2.34KB/s = 162.4$ MIPS servers to guarantee real-time performance. Note that $2.34KB/s$ is the throughput of the average image size in Table 4.11.
2. On the other side, if the number of available servers cannot be changed, the designer can reason on the appropriate image size. If we assume to have 80 MIPS servers in the cluster, then they can process only up to $80 \times 2.34KB/s = 187.2KB/s$. In that case, the average image size must be less than $74KB \times (187.2/380) = 36.5KB$ for the application to work in real-time.
3. The network traffic through the DB server includes picture uploading from mobile phones, picture downloading by the MIPS clusters, updating and reading of geolocation information and people count. Based on the analysis of the network traffic and how it scales as the system grows, the designer can evaluate the best database architecture, e.g., distributed rather than centralized.

Application Design. The application iterates the following work flow:

1. The mobile phone users take pictures and upload then to the Image DB along with their geolocation.
2. The cluster of MIPS servers fetches one image at the time from the DB and counts the people in it, by means of a human recognition algorithm.
3. The number of people in each image is stored back into the DB.
4. The Map Generator creates a plotted image as the result.

Each iteration is done in parallel, in the sense that the multiple Android Emulators upload images and the MIPS servers process the images concurrently. The application consists of the following modules.

Android Camera App. One instance of the Android Camera App runs on each Android Emulator. To simulate the smart phones that users use to take pictures, we took images publicly available on the Picasa and Flickr's image databases [26; 25], and we distributed them across the local storage of the Android Emulators, before starting the simulation. For the experiment, we considered 55,831 images with the geolocation information of Manhattan[9], assuming the users are taking pictures in this area. We modeled the act of a phone user taking a picture with the App loading a picture from the local storage.

Image Database. Every time the App takes a picture, it immediately uploads it, together with its metadata, i.e., latitude and longitude, to the Image DB. Also, the MIPS cluster fetches images and metadata from this database to process them, and stores back the results.

Human Recognition. The human recognition program is based on OpenCV. It is stored in the NETSHIP Server's storage and runs on the MIPS cluster. To detect human bodies in a given picture, we used a head-and-shoulder detecting Haar model [273] and an upper-and-lower-body detecting Haar model [192]. It is, however, difficult to grasp human bodies from multiple directions, in particular from a side view [324].

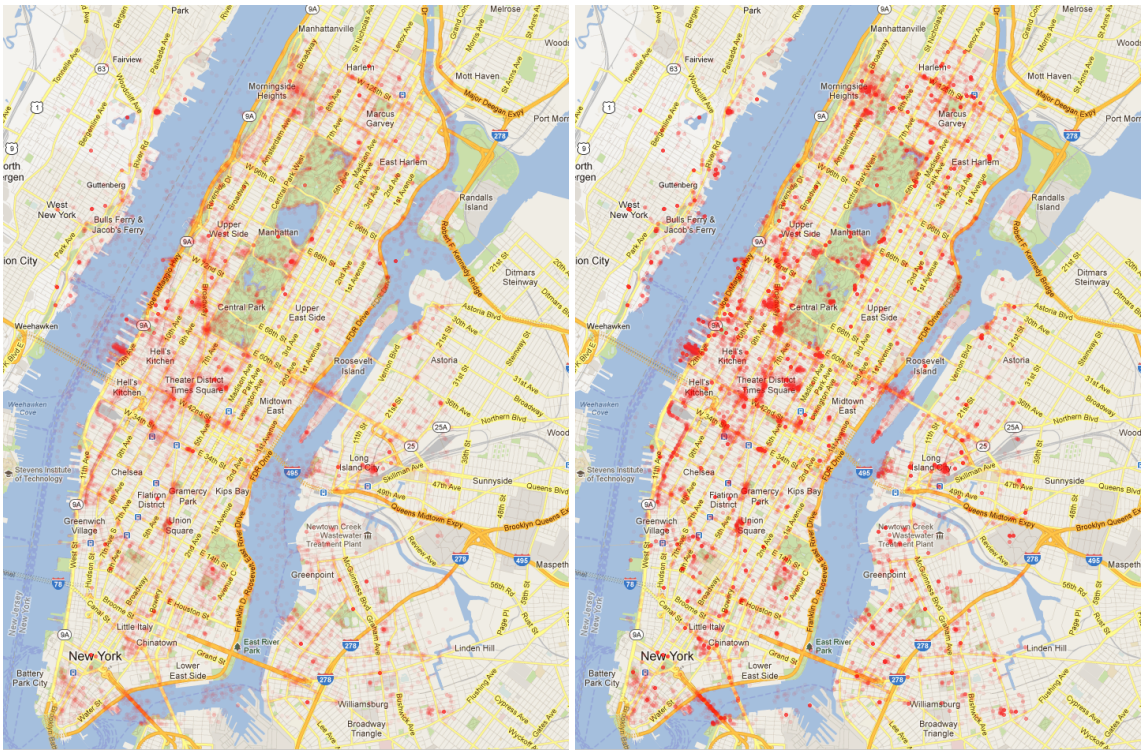
Map Generator. The Map Generator program reads the people counting from the Image DB and plots it on the map[10] with a resolution 717×944, translating latitude and longitude to the pixel position.

Application Results. Although the quality of the developed application's result is not the primary concern of this work, we present the resulted maps from two possible alternative variations of the crowd estimation application: one based on counting only *the number of pictures* taken at a particular location, and the other based on counting *the number of people* showing in those pictures. The results of Figure 4.9 are interesting but they can be substantially improved by using NETSHIP to analyze various possible optimizations of the application.

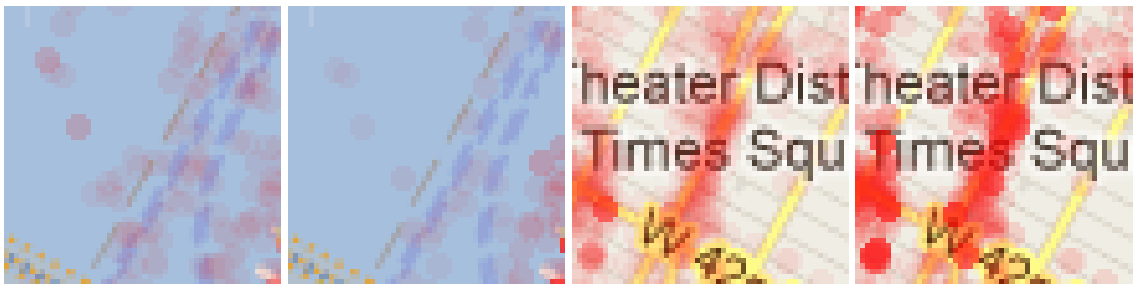
Figure 4.9(a) shows how many pictures are taken by users and Figure 4.9(b) shows the estimated crowds based on such pictures. One red circle on the map corresponds to an area of approximately $2500m^2$ or $2990yd^2$. The density of the crowds is presented with the opacity, where the transparent area indicates no people and an opaque circle indicates more than 80 people in that area.

[9]For the geolocation of Manhattan we used longitude -74.015 $\sim$ -73.928 and latitude 40.700 $\sim$ 40.816.

[10]The map is extracted from the Google Maps service.



(a) Picture Count (b) Crowd Estimates



(c) Tile at (81,395) of (a) (d) Tile at (81,395) of (b) (e) Tile at (210,460) of (a) (f) Tile at (210,460) of (b)

Figure 4.9: Case Study II: visualization of (a) picture count and (b) estimated crowd based on the pictures.

Figure 4.9(c) is a tile taken from Figure 4.9(a) at the pixel position $< 81, 395 >$ and Figure 4.9(d) is a tile taken from Figure 4.9(b) at the same position. Likewise, Figure 4.9(e) is a tile taken from Figure 4.9(a) at the pixel position $< 210, 460 >$ and Figure 4.9(f) is from the same pixel position of Figure 4.9(b).

Both Figure 4.9(a) and Figure 4.9(b) give an idea of which areas are more crowded than others,

based on the opacity of the red circles on the map. However, the comparison of these two figures shows that our crowd estimation algorithm, based on human recognition, gives a more accurate outcome than simply counting the total number of taken pictures. For example, as shown in the comparison of the pair of Figure 4.9(c) and Figure 4.9(d), the estimated crowds on the river was decreased by the human recognition algorithm because the pictures taken over the river are mostly Manhattan skyline photos taken on a boat, a helicopter, or an airplane. On the other hand, in the case of Figure 4.9(e) and Figure 4.9(f), the actual crowds on the ground might be greater than the number of pictures taken on the same spot.

4.7 Related Work

A number of studies have previously focused on helping system architects to better design distributed embedded systems by providing ways to optimize the process scheduling and the communication protocols [157; 253], tools to ease design space explorations [281; 173; 144], estimation models [344], and network behavior simulations [119], or methodologies [193; 144]. Nonetheless, these tools or systems only generate quantitative guidelines that must be then applied to the physical devices, thereby precluding their usability without already having the physical devices in place and the application deployed on them. There are also contributions obtained through the use of VPs [82; 328]; however, none of these works consider the three levels of heterogeneity that characterize more and more distributed embedded systems (Section 4.2). Synchronization between the VP instances, one of the key features of our networked VP, has been inspired by previous works [260; 256]. However, these works do not consider node-level or network-level heterogeneity.

4.8 Concluding Remarks

We have designed and implemented NETSHIP, a framework for building networked VPs that model heterogeneous distributed embedded systems. Networked VPs can be utilized for various purposes, including: i) simulation of distributed applications, ii) systems, power, and performance analysis, and iii) costs modeling and analysis of embedded networks' characteristics.

We also designed hardware accelerators for specific algorithms. We analyzed that accelerators might require more resources of the CPUs that host the simulation. We quantified how this phe-

nomenon partially limits the scalability of the entire networked VP, and provided guidelines on how to distribute the VPs in order to counter balance this loss of simulation performance.

Finally, we used NETSHIP to develop two networked VPs. We used one VP to design a scheduler based on MPI and to verify through simulation how the scheduler is able to optimize the execution of many MPI jobs over a network of heterogeneous machines, by simply distributing the jobs among the available machines on the basis of their performance-per-application profile. We used the other VP to design and validate an application distributed among portable devices and a cloud of servers, and also to derive potential insight about the number of servers and the image size that guarantee the entire application to run in real-time.

Chapter 5

Improving the Design Tool for Mobile GPU Simulations

Despite their proliferation across many embedded platforms, GPUs present still many challenges to embedded-system designers. In particular, GPU-optimized software actually slows down the execution of embedded applications on system simulators. This problem is worse for concurrent simulations of multiple instances of embedded devices equipped with GPUs. To address this challenge, Luca Carloni and I present ΣVP, an extension of NETSHIP, to accelerate concurrent simulations of multiple virtual platforms by leveraging the physical GPUs present on the host machine. ΣVP multiplexes the host GPUs to speed up the concurrent simulations without requiring any change to the original GPU-optimized application code.

With ΣVP, GPU applications run more than 600 times faster than GPU-software emulation on virtual platforms. We also propose Kernel Interleaving and Kernel Coalescing, two techniques that further speed up the simulation by one order of magnitude. Finally, we show how ΣVP supports simulation-based functional validation and performance/power estimation.

5.1 Introduction

Since their advent in 1999, Graphics Processing Units (GPUs) have progressively benefited the performance of many computing systems with their specialized parallel architectures. Originally designed to serve on desktop computers, nowadays GPUs play an important role in a variety of sys-

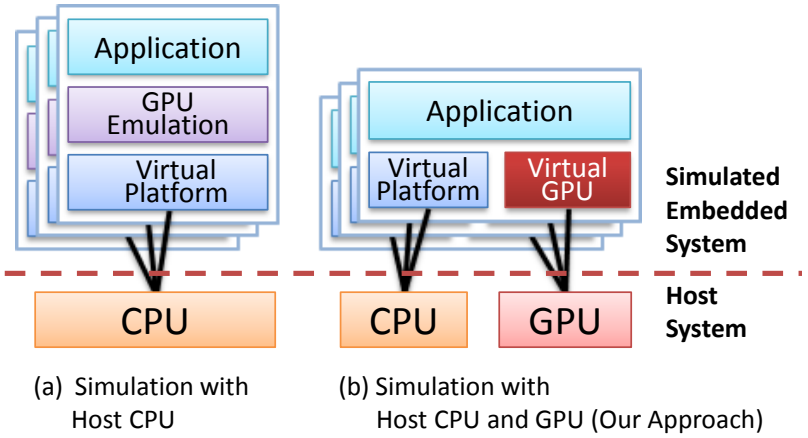


Figure 5.1: Two ways of simulating GPU applications.

tems. Since the introduction of the use of GPUs for general-purpose computing (GPGPU) a growing number of high-performance computing systems have adopted them [242]. GPGPU has found its way also into mobile and embedded systems for a variety of applications, including sensor-data processing and computer vision [137; 337]. Furthermore, these systems are increasingly integrated in large-scale networks to form distributed embedded systems and support such applications as multi-player online gaming [133; 218].

Given these trends, designers are increasingly interested in simulating the execution of GPU applications on the computing systems that they are designing and that will host one or more GPUs. Simulation with multiple instances of virtual platforms (VPs) enables many important design decisions as part of the process of exploring the design space of the *target systems* [168; 167; 178]. Since this process requires the simulation of complex application scenarios, the speed of the simulator is of critical importance.

However, using GPU-optimized code in a simulation environment presents some challenges. While it accelerates a given application on the target system, **the addition of GPU-specialized software code can slow down the simulation** of the application execution. The reason is that most of the current multi-node system simulators run the entire simulation on the *host CPU*. Then, in order to run the GPU code, many simulators, and even widely adopted development tools such as the Android Emulator, need to include GPU emulation capabilities (e.g., the Mesa software backend) [12; 283]. The presence of an additional software layer on top of the VP significantly deteriorates the overall execution speed [162; 318]. Figure 5.1(a) illustrates this scenario of simulating embedded

applications that have GPU code by using GPU emulation on top of a VP that runs on the host CPU. In contrast, we propose to take advantage of the increasing presence of physical GPUs in many host systems. As shown in Figure 5.1(b), the idea is to execute the GPU code from multiple virtual GPU models on the host GPU.

To demonstrate this idea we developed *Simulation using GPU-Multiplexing for Acceleration of Virtual Platforms* (ΣVP), a framework to simulate embedded devices equipped with embedded GPUs. Our system executes separately the target CPU code on the simulated CPU and the target GPU code on the simulated GPU, thus enabling a modular integrated simulation of multiple embedded systems.

ΣVP benefits from two novel optimization techniques, Kernel Interleaving and Kernel Coalescing, that we developed thanks to the possibility of executing multiple VP instances on virtual embedded GPUs. We show that ΣVP can be used for functional validation, timing analysis and estimation of power dissipation. Our approach not only speeds up the simulation time by orders of magnitude but it also enables major savings in terms of the efforts to build the models for these timing and power analyses.

5.2 GPU Multiplexing for Simulation

ΣVP multiplexes the host GPUs to execute the request from the VPs by using separate streams for each VP. Thanks to our methods for time-division multiplexing (interleaved invocations) and throughput-division multiplexing (coalesced invocations), the host GPUs can be used to accelerate the execution of the target GPU code. In this section, we present the components of ΣVP and how they interact.

The Architecture of ΣVP. Figure 5.2 shows the structure of the prototype that we developed to evaluate our ideas. ΣVP supports many VP instances, each consisting of three main modules: a GPU user library, a GPU driver, and a virtual embedded GPU hardware model. We designed these to efficiently resolve the three challenges discussed above.

The **GPU User Library** forms a layer that intercepts the requests from user applications by providing the same APIs of the physical GPUs, e.g., the CUDA runtime library [9]. We designed the user library for the virtual GPU model to support binary compatibility with existing GPU appli-

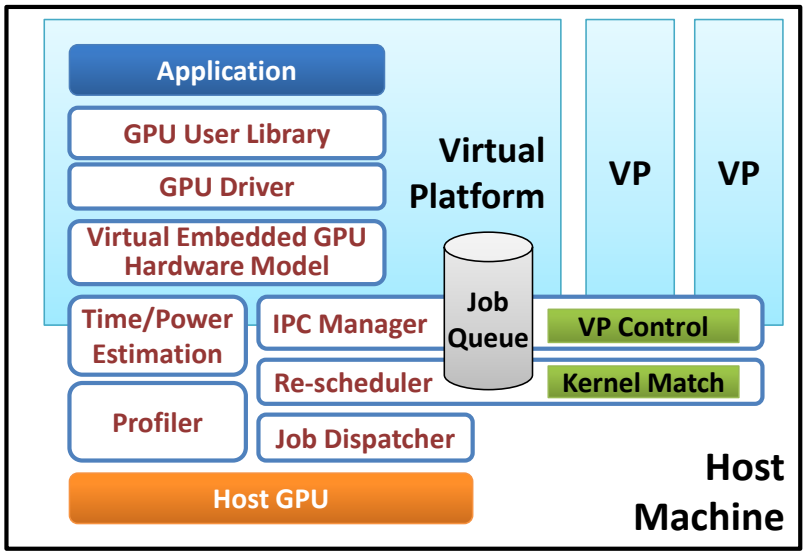


Figure 5.2: Proposed simulation framework prototype.

cations. Hence, the application binaries that use GPU instructions do not need any change to run on the virtual GPUs. Instead, the user library forwards the requests from those applications to the virtual **GPU device driver**. This is a driver for the guest operating system that works as an interface between the GPU user library and the virtual GPU hardware model. Finally, the **Virtual Embedded GPU Hardware Model** pushes the requested kernels into the *Job Queue* in the host machine through the IPC manager.

On the host machine, there are five modules that run on top of the physical GPU. The **Inter-Process Communication (IPC) Manager** allows the virtual embedded GPUs and the host GPU to communicate through an IPC method such as socket or shared memory. Inside the IPC manager, there is a submodule, named VP control, that stops and resumes the VPs to support the Kernel Interleaving optimization technique for synchronous kernel invocations, which is presented in Section 5.3.1. The **Re-scheduler** has two functions. First, it reorders the asynchronous kernel jobs in the Job Queue by keeping a partial order in the original VP. It is a non-preemptive, optimal scheduler augmented for job dependencies [212]. Second, it combines identical kernel requests in the Job Queue into one single kernel job, by using Kernel Coalescing, also discussed in Section 5.3.2. The **Job Dispatcher** links the requests to the GPU driver library on the host machine and invokes the physical GPU instructions based on the requests in the *Job Queue*. The **Time/Power Estimation** module estimates the execution time and the power consumption on the target GPU, while we

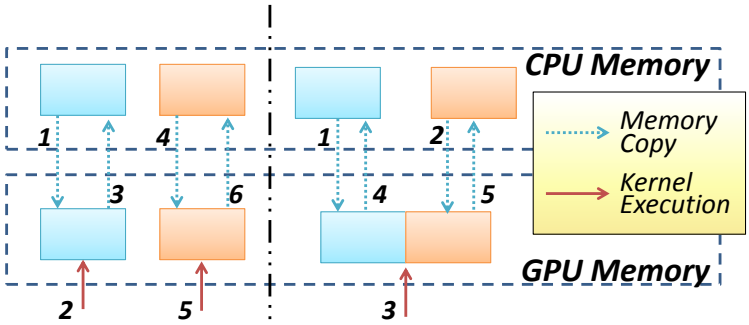


Figure 5.3: Coalescing two memory chunks (left) into one (right).

actually execute the kernel on the host GPU based on the profiling information, as described in Section 5.4. Finally, the **Profiler**, which is provided by the manufacturer, acquires execution information such as the number of executed instructions (per instruction type), the elapsed clock cycles, and the percentages of each occurred stall.

Across the modules of our prototype, it is not necessary to additionally translate the APIs and instructions from the target GPU applications on the virtual platform into other APIs or instructions. This is because most recent GPUs support standard APIs, like OpenGL or OpenCL, in addition to some widely used proprietary APIs such as Microsoft's DirectX [140] and NVIDIA's CUDA [9]. These APIs are available across various vendors and models. Particularly, the APIs supported by embedded GPUs are in many cases compatible with their non-embedded counterparts. For instance, most mobile GPUs nowadays support OpenGL ES, whose APIs are a subset of the standard OpenGL APIs. Also, ordinary GPUs fully support OpenGL.

For the development of our simulation framework prototype we addressed the three main challenges listed above as follows:

1. **Multiplexing**: we used CUDA Streams in the execution process to run the CUDA codes extracted from multiple CUDA programs. By allocating a separate stream to each of the programs' default stream and synchronous APIs, time-division and throughput-division multiplexing become possible. CUDA Context is not a desirable solution here because using multiple contexts introduces significant performance degradation in a process [9]. In our prototype, the *Virtual GPU Model* and *IPC Manager* handle the serialization of the requests while the *Re-scheduler* reorders and merges the requests for multiplexing.
2. **Binary Compatibility**: we used intact CUDA Runtime library on the virtual platforms to convert the CUDA Runtime API requests to CUDA Driver APIs in a similar way as done

by GVirtuS, Barra, and GPU Ocelot [262; 73; 97]. The GPU application binaries compiled for the target architecture are linked to the CUDA Runtime library. By keeping the original CUDA Runtime library, the given application binaries can execute on our simulation framework without the need of recompilation. Only the underlying layers are changed. Thanks to the layered architecture of the CUDA software stack, replacing the CUDA driver library can elegantly address this challenge without incurring any side effects. By delivering extracted PTX codes to the CUDA driver library, this also resolves the problem of extracting PTX instructions. Thus, the *Re-scheduler* can easily identify or modify the kernels in the PTX format.

3. **Communication**: Instead of making a clear-cut decision, we took a hybrid approach by using both socket and shared memory for communication. For short-latency of API invocations, a socket works as a Remote Procedure Call (RPC), while shared memory plays a key role in memory copying triggered by the *cudaMemcpyXXX()* functions.

5.3 Two Optimization Techniques

Kernel Interleaving and Kernel Coalescing are two techniques that we developed to improve the performance of simulating the execution of GPU commands from different applications on multiple virtual-platform instances. Executing multiple virtual platform instances with GPUs simultaneously is obviously a challenge. In the mean time, however, there are opportunities to optimize the executions of GPU commands from different programs. In this section, we propose optimization techniques that efficiently execute multiple GPU kernels. Using these two novel optimization techniques, we improve the execution performance of embedded GPU programs.

5.3.1 Interleaving kernels and memory instructions

GPU architectures feature two types of engines that can operate in parallel: a Compute Engine and a Copy Engine. Many GPU applications, however, iterate over a simple pattern: a memory copy from the host to the GPU device, a kernel execution, and a memory copy from the GPU device to the host. Although some recent GPUs support *Concurrent Kernel Execution* that may automatically interleave kernels from distinct streams, this can lead to suboptimal performance, as

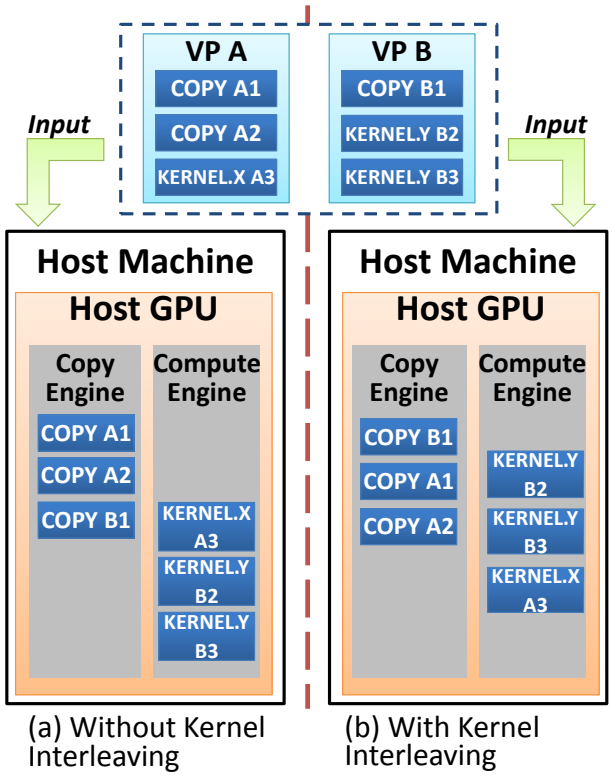
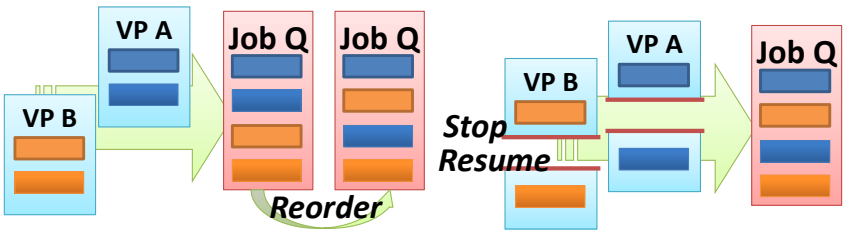


Figure 5.4: Kernel Interleaving.

shown in Figure 5.4(a). Kernel Interleaving reorders the executions to reduce the wasted cycles across the two engines and improves the overall execution time by using the expected time for each invocation, as shown in Figure 5.4(b).

CUDA offers two types of API invocations. *Synchronous invocation* executes API primitives like *cudaMemcpy()* or *cudaLaunch()* and waits until the requested operation is finished before returning to the callers context. *Asynchronous invocation* is possible for user-defined CUDA kernels and some API primitives such as *cudaMemcpyAsync()*: this invocation adds the execution request to a queue and returns immediately to the caller context, while the actual execution will happen in background. Synchronous invocations are easier to program but asynchronous invocations can run faster for some cases.

To implement Kernel Interleaving we followed two distinct approaches for the two kernel-invocation types that are supported by GPUs: synchronous and asynchronous. To effectively interleave instructions from different programs, ΣVP reorders the asynchronous requests in the *Job Queue* as shown in Figure 5.5(a). For synchronous kernel invocations, instead, ΣVP cannot fetch



(a) Reordering forasynchronous requests (b) Interleaving forsynchronous requests

Figure 5.5: Interleaving GPU instructions.

the next GPU instructions until it finishes the current one. However, since it is possible to control the progression of the execution of each VP, we can stop one for some time to let another one run. This property can be used to perform kernel interleaving for synchronous GPU calls, as shown in Figure 5.5(b).

5.3.2 Coalescing Identical Kernels

Generally, the invocation of a function in a program suffers from some overhead: the program must backup and restore the register values, deliver the function arguments, jump to the function code, and finally return to the main program. In many cases of embedded applications, it is important to reduce such overhead. Hence, various programming languages support *inline function expansion*, which allows the compiler to implant the code of a function into the caller code. Function inlining, however, is not available for the invocation of a GPU kernel from a CPU. In this case, the invocation overhead is even larger because it includes the process of sending the request from the CPU to the GPU, a separate device, through a device driver and a hardware channel.

We observed that when multiple VP instances are running in our simulation environment, it is likely that an identical kernel is called by more than one VP at the same time [284]. This occurs, for instance, during the simulation of multiple streaming media boxes, embedded systems which are equipped with a GPU to decode and show video contents to the users [284]. In this case, the boxes are likely to execute the same decoding algorithm at a specific point of time. Such simulations can be accelerated by coalescing those common invocations from each VP into a single kernel invocation. ΣVP makes this possible through an appropriate management of memory. When kernel coalescing is necessary, ΣVP first coalesces the memory chunks into one bigger piece of data stored at physically contiguous memory locations, as shown in Figure 5.3. Then, the GPU can run one

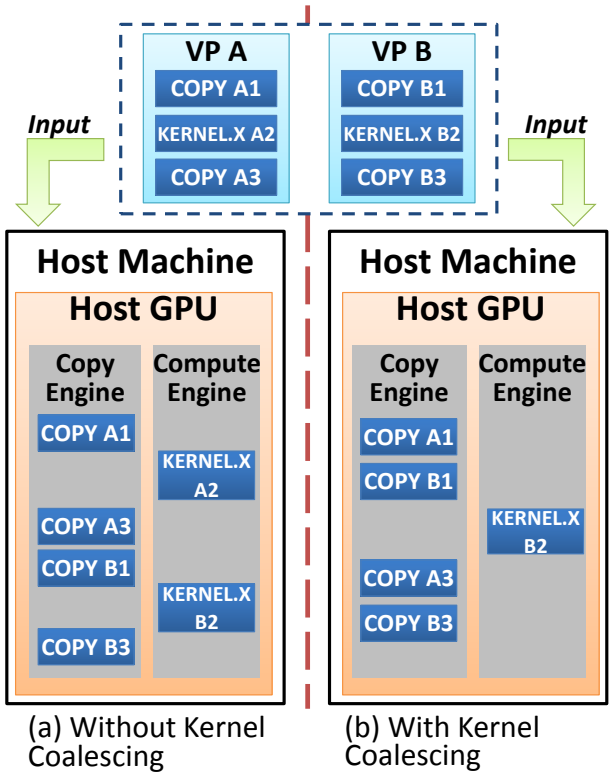


Figure 5.6: Kernel Coalescing.

kernel instance to process the merged data set.[1] After the kernel execution, the resulting data are properly divided to be copied from the GPU device back to the host memory addresses.

Figure 5.6 illustrates this idea for the case of two kernel instances: instead of executing them as shown in Figure 5.6(a), Kernel Coalescing allows us to execute a single kernel instance on a larger data set as shown in Figure 5.6(b).

This technique brings another significant gain: data alignment. Due to their parallel architecture, GPUs are designed to execute multiple concurrent threads. Hence, whenever the data size is not aligned, the GPU must run another loop of the kernel for the rest of the data. This handicap can be significantly reduced by coalescing memory chunks. For example, a GPU with 64 concurrent threads can process 64 data elements simultaneously. Any data set that is a multiple of 64 requires

[1] Merging memory chunks for kernel coalescing is different from global memory access coalescing [183].

```
template <int BLOCK_SIZE> __global__
void matrixMul(float *C, float *A, float *B,
               int wA, int hA, int wB) {

   int bx = blockIdx.x;
   int by = blockIdx.y;
   int tx = threadIdx.x;
   int ty = threadIdx.y;

   // Offset from the coalesced B matrix
   int iy = wA * (int) (by * BLOCK_SIZE / hA);

   int aBegin = wA * BLOCK_SIZE * by;
   int aEnd   = aBegin + wA - 1;
   int aStep  = BLOCK_SIZE;
   int bBegin = BLOCK_SIZE * bx;
   int bStep  = BLOCK_SIZE * wB;

   float Csub = 0;

   for (int a = aBegin, b = bBegin;
        a <= aEnd; a += aStep, b += bStep) {

      __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
      __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

      As[ty][tx] = A[a + wA * ty + tx];
      // Load B matrix from the offset
      Bs[ty][tx] = B[b + wB * (iy + ty) + tx];

      __syncthreads();
#pragma unroll
      for (int k = 0; k < BLOCK_SIZE; ++k) {
          Csub += As[ty][k] * Bs[k][tx];
      }
      __syncthreads();
   }
   int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
   C[c + wB * ty + tx] = Csub;
}
```

Listing 5.1: Coalesceable CUDA kernel for matrix multiplication.

as many thread iterations: e.g., to process 16 memory chunks of the same size, the threads will iterate $3 \times 16 = 48$ times. On the other hand, if this GPU must process 132 data elements, a size that is unaligned with respect to the number of kernel threads, then it has to iterate the 64 threads three times (64 + 64 + 4), the last time for just 4 elements. Coalescing memory chunks can significantly reduce this handicap. For instance, coalescing 16 memory chunks of 132 data elements produces a memory chunk of $16 \times 132 = 2112$ elements, which is processed by iterating the kernel only $2112/64 = 33$ times, instead of 48 times.

Merging Memory Chunks for Kernel Coalescing. Benefiting from coalescing unaligned memory chunks is unattainable for some exceptional kernels that behave differently as the input size changes. Merging memory chunks for Kernel Coalescing requires that the kernel does not access the data across different memory chunks. Matrix multiplication is an example of an application whose memory-access patterns must be modified to make it coalesceable. Listing 5.1 shows an implementation of the matrix multiplication that can benefit from Kernel Coalescing. With respect to the original version provided by NVIDIA, two modifications were necessary (Lines 9 and 26).

5.4 Time and Power Estimation

To augment ΣVP with capabilities for timing and power analysis we developed *Profile-Based Execution Analysis*, a novel method that combines the information obtained executing the kernel on the host GPU with the information obtained compiling it for the target GPU and with existing models for time and power estimation [141; 197].

Figure 5.7 illustrates the main idea of this method. First, ΣVP compiles the kernel for both the target and the host GPU architectures. Second, ΣVP executes the kernel on the host GPU and gathers a variety of kernel-profiling information from this execution including: number of executed instructions for each instruction types (floating point and integer arithmetic, control flow, and memory access), elapsed clock cycles, cache hit/miss counts, and stall reasons. Then, ΣVP derives various execution profiles as if the kernel was executed on the target GPU. For instance, by com-

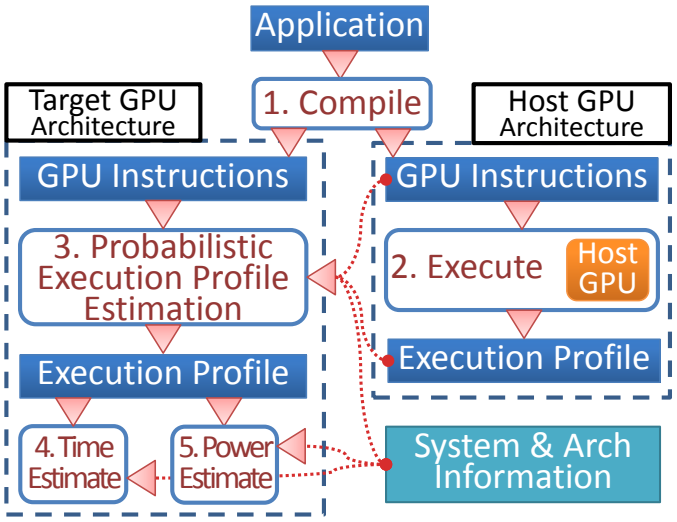


Figure 5.7: Profile-based execution analysis.

bining the iteration count[2] and the number of instructions of each program block[3], ΣVP derives the expected instruction count $\sigma_{\{K,T\}}$ for the kernel K executed on the target GPU T, as shown in Figure 5.8. The same method can be extended to obtain σ for each instruction type i:

$$\sigma_{\{K,T\}} = \sum_{i} \sum_{b} \left[\lambda_b \cdot \mu_{\{b_i,T\}} \right] \tag{5.1}$$

where $b \in K$ is a program block; $i \in \{\text{FP32, FP64, Int, Bit, B, Ld, St}\}$ is an instruction type; $\mu_{\{b,T\}}$ is the static number of instructions from b compiled for T; and λ_b is the iteration count of a block b in the execution.

Timing Estimation. We built three increasingly refined models to estimate the number $C_{\{K,T\}}$ of clock cycles needed to execute a kernel K on the target GPU T. The first model is simply based on $\text{IPC}_{H \to T}$ that is the ratio of IPC_T and IPC_H, which are the maximum values of the number of Instructions Per Cycle on the target (simulated) and the host (simulating) GPU architectures, respectively. Then, we obtain our first estimate $C_{\{K,T\}}$ simply as:

$$C_{\{K,T\}} = \frac{\sigma_{\{K,T\}}}{IPC_H \times IPC_{H \to T}} \tag{5.2}$$

[2]The iteration count can be estimated via several probabilistic methods [83]. For more precise evaluation, we dynamically inserted PTX instructions into the kernel before the execution to obtain the iteration count. This involves less than 0.5% overhead.

[3]The largest portion of the kernel that has a distant execution path determined by control instructions.

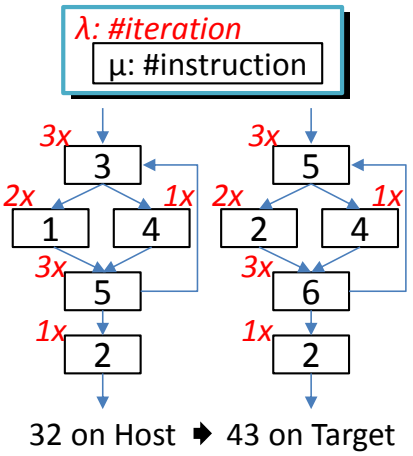


Figure 5.8: Instruction count derivation.

This, however, does not capture the characteristics of each GPU microarchitecture which may have an important impact (e.g., the same instruction may take different clock cycles on different GPUs or a smaller cache size can cause more memory access stalls). To better estimate the IPC_T we can use a probabilistic approach based on the execution latency τ of each instruction type i [197]. Since the ideal number of clock cycles spent on the host (excluding stalls) is given by:

$$C^{\mathfrak{P}}_{\{K,H\}} = \sum_i \left[\sigma_{\{K_i,H\}} \times \tau_{\{i,H\}} \right] \tag{5.3}$$

A second estimate of $C_{\{K,T\}}$ is:

$$C'_{\{K,T\}} = C^{\mathfrak{P}}_{\{K,T\}} + C_{\{K,H\}} - C^{\mathfrak{P}}_{\{K,H\}} \tag{5.4}$$

But this uses the exact stall delays occurred on H, which can lower the estimation accuracy. By augmenting our model with a probabilistic model of the data-cache behavior for data-dependency stalls [255], we get the third estimate:

$$C''_{\{K,T\}} = C'_{\{K,T\}} - \Upsilon^{[data]}_{\{K,H\}} + \Upsilon^{[data]}_{\{K,T\}} \tag{5.5}$$

where $\Upsilon^{[data]}_{\{K,H\}}$ are the data-dependency stalls occurred during the execution of K on H, calculated combining the probabilistic data-cache behavior model and the details of the host GPU architecture (e.g., the main memory size, the cache size and associativity).[4]

[4]Some GPU manufacturers provide the details of their product architectures while some studies discovered the information by microbenchmarking [343].

Power Estimation. Existing power models are based on the number of executed instructions per each instruction type [141]. We use a power-estimation method based on the calculated execution time and the expected execution profile on the target GPU. By combining the power consumption values for each instruction type and the static power dissipation $P_T^{[static]}$, which we empirically acquired, we estimate the power consumption during the execution of K on T as

$$P_{\{K,T\}} = P_T^{[static]} + \sum_i \left[\frac{\sigma_{\{K_i,T\}}}{ET_{\{K,T\}}} \times RP\_Component_{\{i,T\}} \right] \tag{5.6}$$

where $RP\_Component_{\{i,T\}}$ denotes the runtime power consumption dissipated by the microarchitecture components of T to execute the instruction of type i. The estimated execution time $ET_{\{K,T\}}$ is calculated as the estimated clock cycles divided by the product of the number of used GPU processors and the GPU clock frequency. We use C'' as the clock cycles for calculating the estimated power consumption.

5.5 Experimental Setup

In this section, we explain the setup and configurations for the experiments. We used a 32 Intel Xeon CPU machine with a NVIDIA Quadro 4000 GPU as the host environment and a QEMU ARM Versatile PB model as the target simulator. For the experiments of time and power estimation, we used also NVIDIA Grid K520 as another host GPU.

5.5.1 GPU Programming Interface

Our approach for the efficient simulation of multiple embedded systems with GPUs is general and can be applied to environments with different acceleration devices or interfaces like OpenCL. A portable and standard programming framework, OpenCL has significantly improved in its usability, performance, and maturity tools [96]. Hence, it would also be a good candidate to evaluate our methods. For the experiments, however, we chose to use NVIDIA's CUDA as the experimental GPU platform for three main reasons. First, NVIDIA offers a rich set of sample applications and libraries for experiments and practical uses. Second, the popularity of CUDA as a platform for GPGPU application is growing due to its leading performance [35]. Indeed, a fair number of algorithms run faster with CUDA than OpenCL [172]. Finally, thanks to the recent release of the Kayla

CPU		
	Clock	2.66GHz
	Model	Core2 6700
	Arch.	x86 64bit
	# Cores	2
	Cache	4MB
RAM		
	Size	4GB
OS		
	Type	Linux
	Distribution	Ubuntu
	Version	3.5.0-43

Table 5.1: Host Machine Specification.

development platform based on the ARM architecture, CUDA is expected to play an increasing role for embedded systems.

5.5.2 Environments

This section describes the environment configurations used for the experiments in Section 5.6. Table 5.1 shows the specification of the host machine that executes multiple virtual platform instances. The information on the GPU and virtual platform used for our experiments are reported in Tables 5.2 and 5.3, respectively.

5.5.3 Benchmark Applications

For our main set of experiments we use the suite of benchmark applications that is available as part of the CUDA SDK [9]. The following is a brief description of each application:

simpleGL is a user interface (UI) application that modifies vertex positions with CUDA and uses OpenGL to render the geometry.

marchingCubes is a UI application that extracts a geometric iso-surface from a volume dataset using the marching cubes algorithm.

Vendor	NVIDIA
Model	Quadro 4000
CUDA Ver.	5.5
# MultiProc.	8
# CUDA Cores	32
Clock	0.95GHz
L2 Cache Size	512KB
Mem Size	2GB
Mem Clock	1404Mhz
Mem Bandwidth	256-bit
# Copy Engine	2
Texture Align	512

Table 5.2: Host GPU Specification.

Name	QEMU
Ver.	1.0.50
Target	ARM
Model	VersatilPB
Memory Size	1GB
OS	Linux 2.6.32-5

Table 5.3: Virtual Platform Specification.

Mandelbrot computes the Mandelbrot set, a mathematical set of points, and stores the result to a file; the points' boundary is a distinctive two-dimensional fractal shape.

VolumeFilter is a UI application that demonstrates 3D Volumetric Filtering using 3D Textures and 3D Surface Writes.

convolutionSeparable is a separable convolution filter of a 2D signal with a Gaussian kernel; this application writes its result to a file.

dct8x8 runs the Discrete Cosine Transform (DCT) algorithm on a 8×8 pixel frame; this is, by definition, a naive implementation.

recursiveGaussian applies a Gaussian blur to the input image using Deriche's recursive method; its execution time is independent from the filter width; this application writes its result to a file.

SobelFilter is a UI application that performs the Sobel edge detection filter on 8-bit monochrome images and shows the result on the screen.

stereoDisparity computes a stereo disparity map, particularly using CUDA's SIMD SAD (Sum of Absolute Difference) intrinsics, e.g., *vabsdiff*.

binomialOptions is a financial application that evaluates the fair call price for a given set of European options under binomial model.

BlockScholes is a financial application that evaluates the fair call and put prices for a given set of European options by Black-Scholes formula.

MonteCarlo is a financial application that evaluates the fair call price for a given set of European options using the Monte Carlo method.

SobolQRNG generates the Sobol Quasi-random Sequence.

nbody performs a gravitational n-body simulation and presents the result on the screen.

smokeParicles is a UI application that renders smoke particles with volumetric shadows using a half-angle slicing technique.

mergeSort implements a merge sort algorithm.

segmentationTreeThrust demonstrates an approach for the image segmentation tree construction based on Boruvka's *Minimum Spanning Tree* algorithm; this application takes an input image and writes an output image for each segmentation level.

To properly measure the performance of our simulation framework across the benchmark suite, we modified the CUDA applications as follows: 1) while the original applications include a verifi-

Language	Executed by	Time (ms)	Ratio
CUDA	GPU	170.79	1.00
CUDA	Emul. on CPU	9141.51	53.52
CUDA	Emul. on VP	374534.34	2192.95
CUDA	**This work**	568.12	3.32
C	CPU	8213.09	48.09
C	VP	269874.03	1580.15

Table 5.4: Execution time of matrix multiplication.

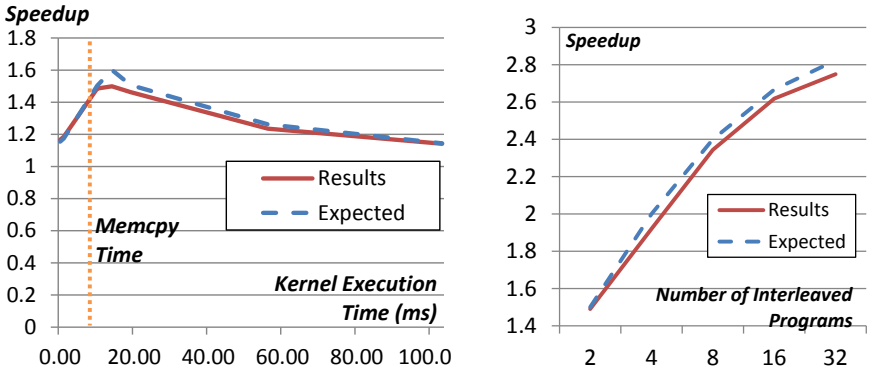
cation phase to compare the outcome from GPU computation with the expected result, we excluded the time spent in this phase during performance measurement; 2) in those cases when the execution time of an application with the default input is too short, we either increased the size of the input or modified the application to repeat the core execution multiple times, in order to make it comparable to the other applications; 3) those applications that require a user screen to present graphic output were configured to use a virtual screen provided by the virtual platform; and 4) applications that run forever (e.g., interactive application that continuously wait for a new input from the user) were modified to run for a few hundreds to thousands loops and then terminate.

5.6 Experimental Results

In this section we present a comprehensive set of experimental results that demonstrate the effectiveness of each of the methods described in the previous section.

5.6.1 Leveraging host GPU for mobile GPU simulations

Our first experiment was a comparative evaluation of the different options to execute an embedded GPU application. For this we used a simple program that multiplies 300 times two 320×320 matrices of double-precision numbers. Table 5.4 reports the results for two versions of the program: a CUDA implementation (first 3 rows) and a C implementation (last 2 rows.) The first row, which corresponds to the native execution of the CUDA program on a GPU, is used as the baseline for the comparison. The execution of the CUDA program takes 53.52 times longer when running on a GPU emulator on top of a CPU and 2200 times longer when running on an ARM CPU model



(a) Speedups for Kernel Lengths (b) Speedups for VP Numbers

Figure 5.9: Experiments for Kernel Interleaving.

inside a VP through binary translation. Clearly, emulating GPU code inside a virtualized emulation model yields suboptimal results. Nevertheless, this is still a common practice in many simulation frameworks of commercial products, which, for instance, use the MESA open-source libraries to run OpenGL ES applications [162]. In fact, as shown by the last two rows of Table 5.4, running the C version of this program on either the CPU or the VP is faster than running the CUDA program on a GPU emulator inside a VP. In contrast, our proposed GPU multiplexing technique is only 3.32 times slower than native execution.

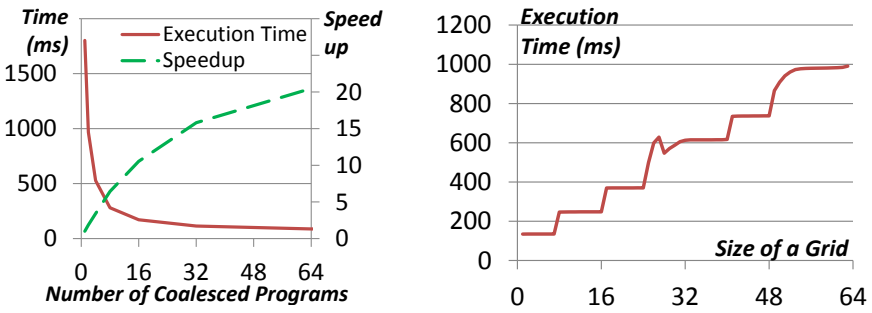
Kernel Interleaving. We consider two interleaved GPU programs, each with a loop that iterates: a memory copy from host to device, a kernel execution, and a memory copy from device to host. Figure 5.9(a) shows the speedups measured as varying the complexity of the kernel while keeping the size of the input data constant. The time for memory copy is 13.44 ms, represented as a vertical orange dotted line.

Kernel Interleaving can shorten the total execution time from $3N$ instructions to $2 + N$ instructions, where N is the number of programs to be interleaved, under the assumption that each instruction takes about the same amount of time. If the kernel execution time T_k and memory copy time [5] T_m are different, the total time is given by:

$$T_{total} = 2T_m + N \cdot Max(T_m, T_k) \tag{5.7}$$

which is represented by the blue line in Figure 5.9(a). The red line shows the actually measured

[5]This means the time for memory copies before the kernel execution; e.g., matrix multiplication needs two input memory copies.



(a) Coalescence Effectiveness (b) Impact of Data Size

Figure 5.10: Experiments for Kernel Coalescing.

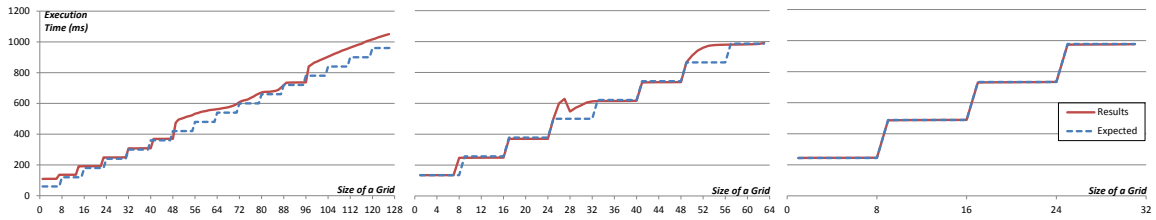
experimental values, which are quite close to the expected values. This experiment confirms that the highest speedup through Kernel Interleaving is obtained when the kernel execution time is similar to the memory copy time (indicated by the orange dotted line). This is a form of latency hiding.

While the previous experiment is for two interleaved programs, Figure 5.9(b) shows the speedups as function of N interleaved programs, from 2 to 32. Since the execution time without Kernel Interleaving is $3T$ when $T_k = T_m = T$, the speedup is expected to grow with N as:

$$Speedup = \frac{3 \cdot N \cdot T}{(2+N) \cdot T} = \frac{3N}{2+N} \tag{5.8}$$

which is represented by the blue line in Figure 5.9(b). For large number of interleaved programs the speedup is about $3\times$.

Kernel Coalescing. Figure 5.10(a) shows the speed of executing *vectorAdd* as function of the number of GPU programs to coalesce. The total size of the input vectors remains the same across the different numbers of programs. In other words, the same amount of work is distributed over the given number of programs. The solid red line indicates the total execution time of the coalesced program and the green dashed line indicates the speedup of the same horizontal coordination, with the result of no coalescing (one program) being the comparison base. For instance, when coalescing 16 GPU programs the time to complete the executions of the applications is 171 ms, for a 10.54X speedup. These results confirm that Kernel Coalescing can indeed reduce the execution time. The speedup reaches 20.48 times for the case of 64 programs. A large portion of the gain can be attributed to the impact of data-size alignment given the number of concurrent threads, the unit of computation the GPU can simultaneously hold. In CUDA the number of concurrent threads used for a kernel is decided by the size of a block (a group of threads) and the size of a grid (a group of



(a) Block size = 256 (b) Block size = 512 (c) Block size = 1024

Figure 5.11: Kernel execution time as function of the grid size and block size.

blocks). A kernel is executed by a grid of thread blocks. Figure 5.10(b) shows the execution time of one single kernel as the size of the data grows and, accordingly, the size of a grid increases from 1 to 64 (while the number of threads in a block remains 512). The resulting curve roughly resembles a staircase, which implies that a kernel execution with an unaligned grid size wastes some portion of its resources. For instance, the same execution time is obtained both for a grid of size 9 and a grid of size 16 even though the data sizes to be processed are different, being $9 \times 512 = 4608$ and $16 \times 512 = 8192$ data units, respectively. For a given size of a grid the expected execution time is

$$T_{expect} = T_o + T_e \times \lceil \xi_{input} / \lambda \rceil \tag{5.9}$$

where T_o is the overhead time spent for launching kernels, T_e is the kernel execution time for the alignment unit size of data, ξ_{input} is the size of the input data, and λ is the aligned unit for the GPU's processing ability.

In summary, these preliminary experiments confirm the effectiveness of the two optimization techniques that we developed in ΣVP so that they can be automatically applied to the simulation of embedded GPU programs on VPs.

5.6.2 Impact of Alignment on Kernel Performance

Recall that the data size is the product of the size of the grid times the block size. In order to better understand the impact of the data size on the execution we run an additional experiment to break down the contributions of the grid size versus the block size. Fig. 5.11 shows the execution times of a simple vector-addition kernel as function of the grid size, varying from 1 to 64, and the block size, varying from 256 to 1024. In these figures, the blue dotted lines are the expected values, calculated with Equation (5.9). Across the different number of threads in a block, the graphs indicate that

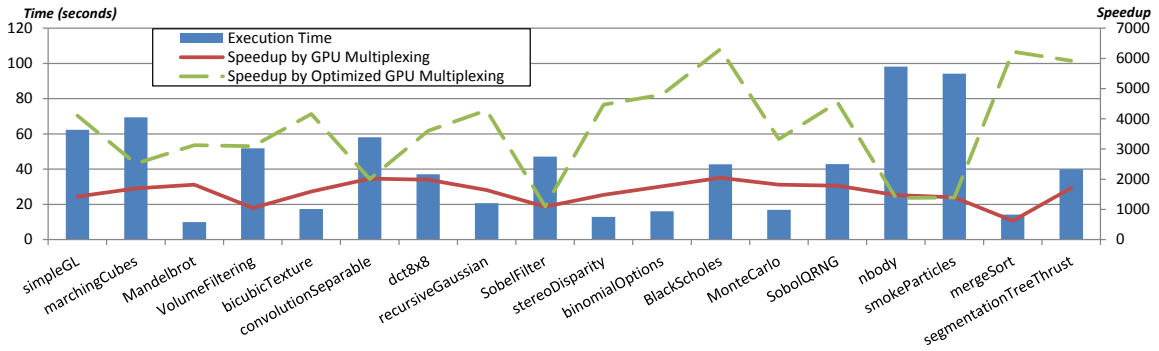


Figure 5.12: Experimental results: GPU-VP emulation vs ΣVP with optimizations.

the execution time is affected more by the alignment of the grid size than by the size of the data. Common across the three graphs is the staircase pattern independently from the block size: every time the grid size increases by eight the latency goes up.

Overall, the measured values track the expected values quite well, particularly for the results of Fig. 5.11(c). Some irregularities can be noticed for the values of the block size equal to 256 and 512 in Fig. 5.11(a) and 5.11(b), respectively. In particular, Fig. 5.11(b) shows a *performance scaling inversion*, where the execution time for a smaller data size is actually longer. The execution time is 628.909 for data size equal to 13824 (512×27), while it is 546.73 for data size equal to 14336 (512×28). We believe that this is an effect of the particular GPU implementation that we used, caused by hardware alignment and optimization. This sort of experimental profiling allows us to avoid the penalty for the specific data size, e.g., by increasing the grid size to be equal to 28 when the requested grid size is 27.

5.6.3 Performance Comparisons

Here we present a complete evaluation of our simulation framework using the suite of benchmark GPU applications available as part of the CUDA SDK [9]. In particular, we compared the simulation of these applications on the VPs for three scenarios: 1) GPU emulation on the VP; 2) simulation on the host GPU with our proposed GPU multiplexing; and 3) simulation on the host GPU with our GPU multiplexing plus the two optimization techniques: Kernel Interleaving and Kernel Coalescing.

Figure 5.12 reports the experimental results. The blue bar shows the execution time of emulating the GPU applications concurrently on eight VP instances. For example, when each of these executes

simpleGL, the time for completing all the executions is about 62 seconds. The green dashed line and the red solid line indicate the speedup achieved by the host GPU multiplexing with and without the two proposed optimization techniques, respectively. Thus, for *simpleGL* GPU multiplexing provides a simulation speedup of 1428 (with respect to the blue bar), while the addition of the two optimizations achieves a speedup equal to 4104.

The analysis of the red solid line suggests that applications that use less floating-point instructions, e.g., *VolumeFilter*, *SobelFilter*, *stereoDisparity*, and *mergeSort*, have relatively lower speedups than others. Also, some non-CUDA operations (e.g., file operations or OpenGL invocations) limit the speedups for *Mandelbrot*, *bicubicTexture*, *recursiveGaussian*, *MonteCarlo*, and *segmentationTreeThrust*, which read from input files or write to output files, as well as *simpleGL*, *marchingCubes*, *VolumeFiltering*, *SobelFilter*, *nbody*, and *smokeParticles*, which use OpenGL for graphics. The reason is that these portions of the applications are not the target of the acceleration provided by ΣVP.

The analysis of the green dashed line confirms that the effect of the two optimization techniques varies across the applications based on their use of CUDA instructions. Some applications like *convolutionSeparable*, *dct8x8*, *SobelFilter*, *MonteCarlo*, *nbody*, and *smokeParticles*, have kernels that cannot be coalesced, thereby intrinsically limiting the achievable speedup. Other applications, like *simpleGL*, *dct8x8*, *BlackScholes*, *MonteCarlo*, *mergeSort*, and *segmentationTreeThrust* copy a memory block from host to device before the execution of a kernel, while the rest of the programs in the experiments generate their own input or copy initial data only. In these cases, the speedup is not as high as when both optimizations are applicable.

Overall, the experimental results are very positive. The speedup obtained with GPU multiplexing varies from 622 times (*mergeSort*) to 2045 times (*BlackScholes*) compared to the emulated GPU on the VP. The speedup with both GPU multiplexing and the two optimizations varies between 1098 times (*SobelFilter*) and 6304 times (*BlackScholes*). In the best case (*mergeSort*) the addition of the two optimizations yields an additional 10X speedup.

5.6.4 Graphical Outcomes from GPU Applications

To validate the correctness of the simulation framework implementation, we compared the results from the executions of GPU applications on the virtual platforms without using host GPUs with

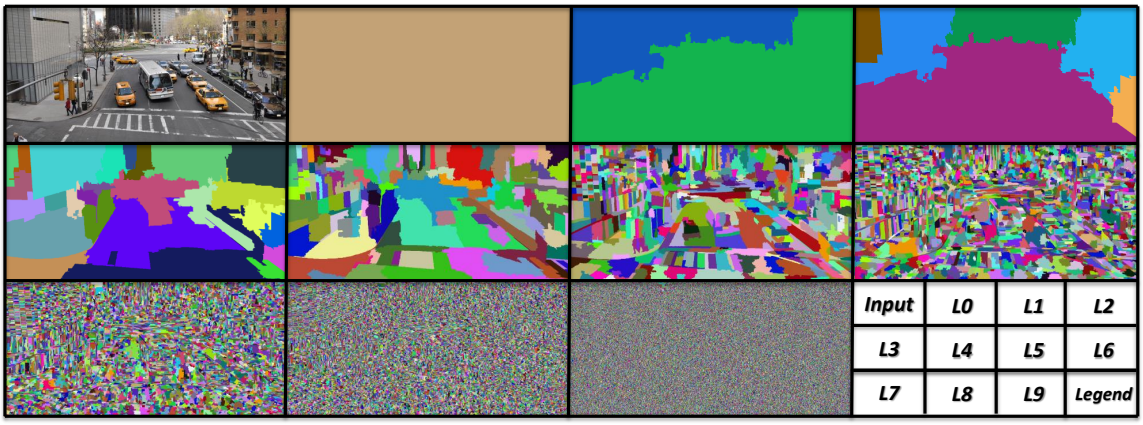


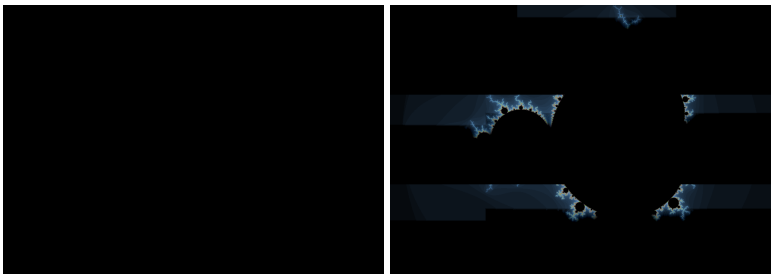
Figure 5.13: An input image and its segmentation results.

the results obtained using host GPUs. To visually present this analysis we report the outcome from *segmentationTreeThrust*. Fig. 5.13 shows the resulted images from each of the ten levels determined by the algorithm. As for all other experiments, we used eight virtual platform instances to execute the same application. Then, we collected one or two images from each virtual platform. The results are correct. They pass data comparison with an error rate under 0.15%.

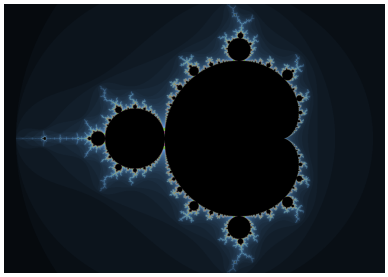
Fig. 5.14 shows three results from *Mandelbrot*. First, we modified the application to distribute its computation and image-generation across the eight virtual platforms. Second, we also modified it to stop its kernel execution and dump its current result after the given execution time. The three images in Fig. 5.14 show the results of executing this application for the same period of time on the three different configurations which we used for the experiments of Fig. 5.12: GPU emulation, GPU multiplexing, and GPU multiplexing with optimizations. Only the configuration using both GPU multiplexing and our two proposed optimization completes the execution in the allotted time, thus returning the complete image, as shown in Fig. 5.14(c).

5.6.5 Timing Estimation

We evaluated the accuracy of our timing estimation models as follows. The estimated time values are calculated for the target GPU (NVIDIA Tegra K1) and normalized by the observed execution time on an actual target GPU. We experiment with execution profile from two different host GPUs, NVIDIA Quadro 4000 and Grid K520. Figure 5.15 shows the measured execution times on the



(a) GPU Emulation (b) GPU Multiplexing



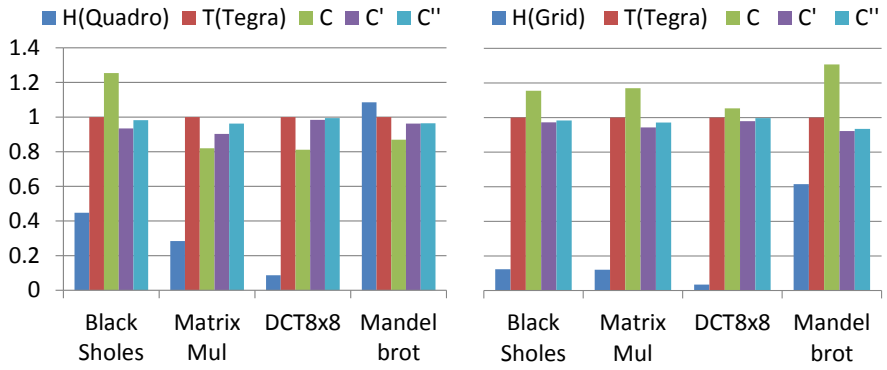
(c) With Optimizations

Figure 5.14: Experimental results: Mandelbrot executed for the same time period.

target GPU and the host GPU, and the three expected execution times $ET_{\{K,T\}}$ based on C, C', and C'', respectively. As expected, the execution times observed on the host GPU are much shorter than the observed and estimated values for the target GPU. On the other hand, the results demonstrate that the estimated execution times are close to the measured values from a real target device. The fact that the estimates are close to 1 no matter which host GPU is used for execution profile confirms that our models work well across the different host GPU architectures.

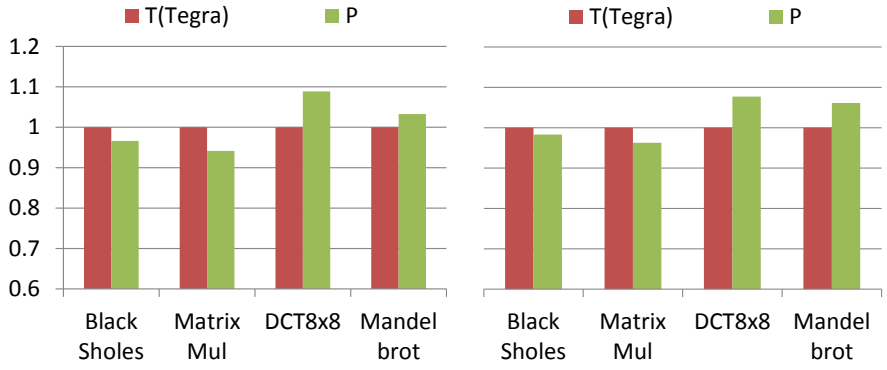
5.6.6 Power Estimation

The results of Figure 5.16 compare the estimated power dissipation with the one measured on the actual device. Our estimations are within about 10% of the actual values, thus confirming that ΣVP can be effectively used also for simulation-driven power analysis.



(a) Estimates using Quadro 4000 (b) Estimates using Grid K520

Figure 5.15: Normalized execution times: two observations on target and host GPUs and three estimates.



(a) Estimates using Quadro 4000 (b) Estimates using Grid K520

Figure 5.16: Normalized power dissipation: an observation on target GPU and an estimate $P_{\{K,T\}}$.

5.7 Related Work

A recent Android platform's experimental patch brings the OpenGL ES 2.0 instructions from the emulator to the host OS, converts to standard OpenGL 2.0, and runs natively on the host GPU [12]. This approach, however, works only for OpenGL in a single emulator. Many GPU simulators are based on software models for GPUs [73]. Some timing estimation methods use these software models to obtain execution traces [197]. These approaches run very slow, while ΣVP offers effective ways to estimate execution time and power consumption in addition to fast simulation.

5.8 Limitations & Future Work

Merging memory chunks for Kernel Coalescing may not work correctly for some kernels that use global variables or access the entire memory span. This forced us to modify some applications for our experiments. This modification has been discussed in Section 5.3.2 for the case of matrix multiplication. The automation of this process using compilation technology is a promising avenue of future research.

In building our simulation framework, we created a virtual embedded GPU model which can be extended by implementing a timing model of the target GPU to support various analysis and experiments like modeling of power consumption and cycle-accurate performance. Although NVIDIA began to support the ARM platforms with CUDA 5.5 SDK, it works only on the *ARM Hard Float* architecture, i.e., ARMv7 with specific floating-point hardware extension, and on Debian-affiliated operating systems. Therefore, an interesting open question is how to build a time-accurate performance model for CUDA integrating virtual platform models without the hard float functionality.

5.9 Concluding Remarks

We proposed a technique to efficiently simulate multiple instances of virtual platforms that run GPU applications. Compared to the emulation of GPUs on VPs, the speed of our simulation framework is between 1000 and 6000 times faster when running a large set of GPU applications. We achieved this major improvement by leveraging the presence of GPUs on the host systems and by optimizing the execution of GPU kernels with two novel optimization methods: Kernel Interleaving and Kernel Coalescing. Further, by presenting a novel estimation method that leverages the execution of a kernel on the host GPU, we showed how our framework can be used not only for full-system simulation but also for timing analysis and power estimation.

Chapter 6

Extending the Design Tool for IoT-Integrated Systems

Internet-of-Things (IoT) is a scenario where objects, including people devices, home appliances, and animals, are identified by unique IDs and communicate over networks. Although the concept has emerged over a decade ago, the combination of wireless networks, sensor technologies, big data processing, and the Internet has brought the recent boom of IoT. Nowadays, many companies invest in the IoT market, developing new IoT systems and applications. Since the size of the systems is enormous and their architectures are highly heterogeneous, design and development of IoT systems is a complex, time-consuming work. Nonetheless, there are no appropriate design tools that can help the designers of these large-scale heterogeneous systems and, particularly, that enable comprehensive exploration of their design spaces.

Together with Prof. Luca Carloni, I propose SimbIoTics, an extension of NETSHIP, to support the design and simulation of IoT systems where millions of IoT nodes interact closely. These nodes may include unmanned aerial vehicles, embedded systems, cloud server computers, and wireless sensors. To realize SimbIoTics, we integrated wireless sensor simulators and unmanned aerial vehicle simulators into NETSHIP. We also invented a highly efficient synchronization method for simulation scalability.

6.1 Introduction

As an increasing number of information-technology services are expanding their markets worldwide, many classes of computing systems reach a global scale, implemented across multiple remote data centers [299]. These systems are also evolving in a way that incorporates a variety of heterogeneous components. In the design of computing systems, increased scalability and heterogeneity typically leads to an increase in design complexity.

The number of design choices for software and hardware, such as distribution of computations, number of computing server nodes, or adoption of hardware accelerators, becomes very large. Thus, the engineering cost of evaluating each option can become significantly high. Recent simulation tools that support the design of large-scale, heterogeneous distributed systems try to address this issue by supporting some degree of *Design Space Exploration* (DSE) [167; 276].

The idea of Internet-of-Things (IoT) anticipates a world where everything that is electronically operated (home appliances, sensors, or outdoor stationary devices, etc.) will be interconnected. Indeed, IoT is expected to spread to our daily lives, allowing connections between objects and humans [121]. Hence, the scale of the IoT-integrated systems will grow exponentially, far beyond the scale achieved by current large-scale computing systems. Likewise, the span of heterogeneity in IoT systems will be much larger due the variety of form factors, networks, computing cores, or peripherals devices. In this context, there are not many simulation tools that are available for system designers to evaluate possible design choices for heterogeneous computing systems where cloud server instances, stationary embedded computers, smart devices, vehicles, and IoT devices operate interactively.

In this chapter, we propose SimbIoTics, a simulation framework for IoT-integrated systems to address the difficulties of designing large-scale, heterogeneous computing systems that comprise many IoT devices. SimbIoTics efficiently simulates concurrent executions of more than hundreds of thousands of instances of cloud server programs, embedded software for stationary systems, mobile apps for smart devices, and wireless sensor applications. Our contributions include: 1) the first simulation tool supporting the modeling of IoT systems with such a large number of heterogeneous components, including embedded systems, wireless sensors, and cloud servers; 2) a novel synchronization method based on the data-dependencies of IoT nodes; and 3) the application of SimbIoTics to the design of a complex IoT system, a traffic management system (TMS) for metropolitan area.

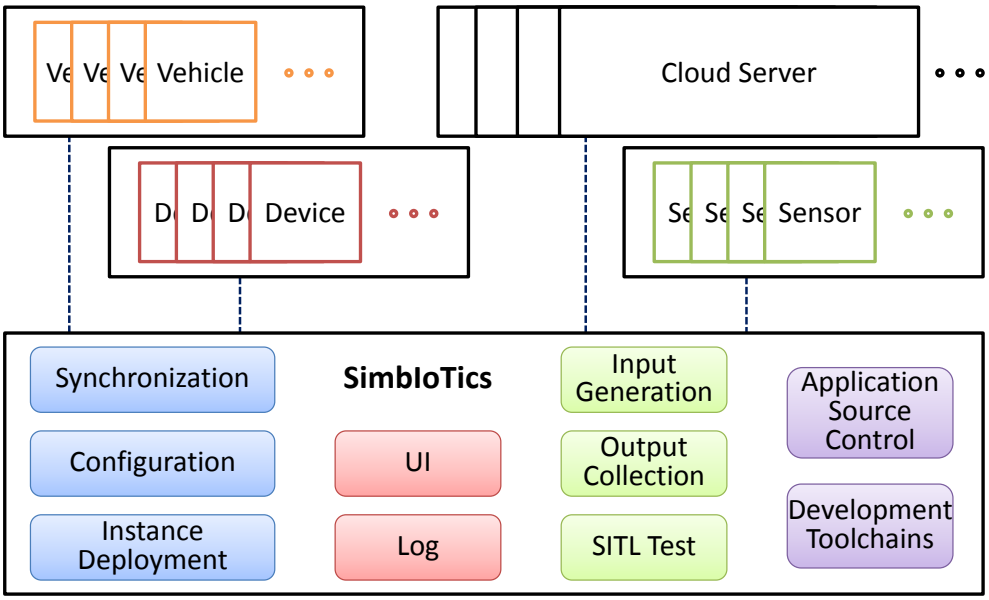


Figure 6.1: The architecture of SimbIoTics.

6.2 Design of SimbIoTics

In this section, we explain how we built SimbIoTics by combining existing simulators and developing new mechanisms for scalable synchronization among them.

6.2.1 Architecture

Figure 6.1 presents the architecture of SimbIoTics. SymbIoTics uses multiple platform (VP) instances for the simulation of the target embedded systems [28]. We run VP instances and other simulators on virtual machine (VM) instances to take advantage of the VP-on-VM model proposed by Jung et al. [167]. SimbIoTics requires two or more VM instances, one for running the SimbIoTics server and other instances each of which executes up to thousands of VPs, wireless sensor simulators, or UAV simulators. We configured some VM instances to simulate cloud servers in the target system. The **synchronization** module makes these distinct simulation environments run with synchronized simulation times using the method proposed in Section 6.2.3, thus ensuring the correctness of the simulation results. The **configuration** module stores and provides the setup values for the VM instance deployment, the system simulator installation, the network configurations, and the links for input and output. The **instance deployment** module scales the simulation environment up or down according to the configuration module. The **UI** module is the interface for the users to

control SimbIoTics, to adjust the configurations, and to check the results of the simulations. The **log** module stores the logging information generated during the simulations. The **input generation** module feeds the input data into SimbIoTics by fetching, creating, parsing, or reorganizing. The **output collection** module collects the results from each simulator involved in the entire simulation. The **Software-in-the-Loop (SITL) test** module enables the deployment of a newly developed component within the simulation and an easy performance evaluation of the component. The **application source control** module manages the application source programs that run on the different simulators using a source control program, *git*, for the heterogeneous execution environments. The **development toolchains** involves the toolkits for developing, cross-compiling and debugging the distinct simulation environments, including heterogeneous processing cores and various operating systems.

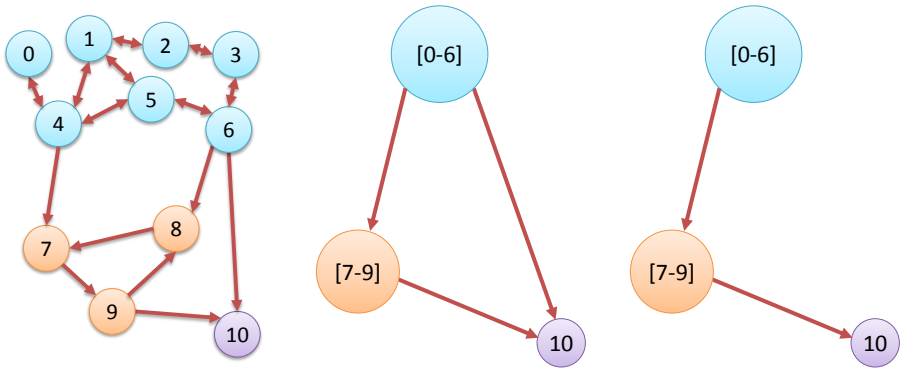
6.2.2 Heterogeneity

IoT systems are characterized by a large variety in terms of the types of the computing nodes and their characteristics, including: form factors, processing cores, networks, computing resources, energy budgets, or peripherals. To simulate the heterogeneous IoT systems, SimbIoTics combines several distinct simulation engines for different types of system components.

Cloud server simulation. SimbIoTics runs on the cloud leveraging the flexibility offered by the cloud's Infrastructure-as-a-Service (IaaS) model. IoT systems comprise cloud servers to store and process the data collected from various sensor nodes. SimbIoTics can dynamically add VM instances to simulate the cloud server machines in the IoT systems.

Embedded system simulation. SimbIoTics uses VPs, software models to simulate target embedded hardware devices that can execute the target software program through dynamic binary translation. These VPs vary in their processing cores and software stacks due to the different types of stationary systems, mobile devices, and vehicles. We relieved the difficulties of integrating VPs and SimbIoTics by designing and implementing a unified API layer that encapsulates the heterogeneous VPs.

Wireless sensor simulation. Although wireless sensors can be seen as embedded systems, they have some distinct characteristics that make using a separate simulator for wireless sensors more efficient and accurate. For example, each sensor has a low-power processor that runs a light-



(a) G: Dependency (b) G': Condensed (c) G": T-Reduced

Figure 6.2: A data-dependency graph across the distributed nodes being processed for simulation synchronization.

weight software program. Also, wireless network protocols are better simulated with simulators that are specifically designed for wireless sensors. SimbIoTics integrates Cooja [94], a simulator for wireless sensor networks to accurately simulate IoT systems that rely on the data from a number of wireless sensors.

Unmanned Aerial Vehicle (UAV) simulation. Certain IoT systems, such as a TMS, can use UAVs for images taken from the sky, enabling image processing for tracking pedestrians, vehicles, and so on. While many parts of UAV systems can be simulated by VPs, using UAV simulators allows us to use various existing dynamic aircraft behavior models. SimbIoTics enables the simulation of complex IoT systems that involve UAVs by integrating the ArduPilot simulator [206], an open source software program to simulate autopilot UAVs.

6.2.3 Scalability

The scale of IoT systems can easily exceed the scale of other types of distributed systems due to the huge number of 'things' that can be integrated in an IoT system. In this section, we revisit the synchronization technique that we introduced in Chapter 4 and propose a new method to significantly reduce synchronization time, the most critical overhead in scaling the simulation of IoT systems.

Traditional Synchronization (TS). A distributed environment that integrates separate simulation processes, such as NETSHIP, requires simulation time synchronization. In such a system, there are multiple processes each of which simulates a node in the target system. These individual

processes can advance their own simulation time differently without synchronizing their simulation times as introduced in Section 4.2. For instance, this allows a node with a past simulation time (running slow) and a node with a future simulation time (running fast) to communicate, which normally does not happen in the real world.[1] To address this problem, NETSHIP stops the nodes whose simulation times are faster than the slowest one until all the nodes have simulation times that are within the pre-configured value Δt. The simulation time Sim_t of each node follows:

$$\text{Sim}_t \leq \text{Sim}_t(N_i) < \text{Sim}_t + \Delta t \tag{6.1}$$

where N_i is a node in the system and $\text{Sim}_t(N_i)$ presents the current simulation time for N_i. When every node reaches $\text{Sim}_t + \Delta t$, $\text{Sim}_t(N)$ is increased by Δt. Although some previous efforts have reduced the overhead for synchronization in the simulation of distributed systems [82; 167], they are not efficient enough for the unprecedented scale of IoT system simulations.

Loosely Chasing Synchronization. To decrease the overhead in synchronization, we propose the *Loosely Chasing Synchronization* (LCS) method that groups nodes and then synchronizes only: 1) nodes within each group and 2) groups that have a dependency relation. Instead of synchronizing all the nodes in the system at each synchronizations step, LCS reduces a large portion of the efforts and overhead as it synchronizes only an essential subset of nodes. Let an *independent* node A generate some data and a *dependent* node B use the data. By keeping A's simulation time more advanced than B's, LCS ensures that the data generated by A is always ready when B accesses it. A, however, does not need to wait for B. [2] LCS is based on the following steps:

1. Draw a directed graph $G = (V, E)$ of distributed nodes with data-dependency among them as shown in Figure 6.2(a). A node represents a vertex and a dependency between two nodes represents an edge between the two corresponding vertices. Synchronous communication methods that affect the behavior of both sides such as socket communications are considered as bidirectional dependencies. Asynchronous communication methods, like buffered messages or file creation, correspond to single directional dependencies.

2. Derive a graph G' by condensation, e.g., by contracting each strongly connected component into one vertex. In G', some vertices (nodes) are grouped into one vertex (a node group) as

[1] We assume that time traveling messaging is not available in the target system.

[2] We avoid overwriting by timestamp the data.

shown in Figure 6.2(b). G' becomes a Directed Acyclic Graph (DAG) by the definition of condensation [160].

3. Find a transitive reduction $G'' = (V'', E'')$ of G' as shown in Figure 6.2(c). The synchronization overhead is reduced by the removed edge in the figure. A transitive reduction of a DAG corresponds to the minimum equivalent graph of the DAG.

4. Simulate with synchronization. All the nodes $i \in V$ that belong to the corresponding condensed group $j \in V''$ need to be synchronized to a simulation time maintained for j as follows:

$$\text{Sim}_t(j) \leq \text{Sim}_t(i) < \text{Sim}_t(j) + \Delta t_j \tag{6.2}$$

Δt can vary according to the characteristics of the node group. A large synchronization time step, e.g., 20 seconds, can work well for time-insensitive nodes such as asynchronous batch processing servers while time sensitive nodes, such as self-driving vehicles communicating with sensors, may need to be synchronized more frequently, e.g., every 500 ms. The simulation times of two node groups, $x \in V''$ and $y \in V''$, need to be one-directionally synchronized if there is a dependency between them as follows:

$$\text{Sim}_t(x) \geq \text{Sim}_t(y),\ \exists (x, y) \in E'' \tag{6.3}$$

This ensures that the data generating nodes are simulated at a speed that is no slower than the dependent nodes which use the data.

While still effectively preventing synchronization problems, this method significantly reduces the overhead in synchronizing the simulated nodes compared to the existing method, i.e., synchronizing all the nodes, due to the following reasons.

First, the method reduces the number of nodes that have to be kept synchronized under a single simulation time. This exponentially decreases the probability of delaying all the nodes in the group. For example, if the probability that a node can run slower (thus delaying other nodes) in a step is 1% in a simulation of 20 nodes, the probability that the entire simulation is delayed is $1 - 0.99^{20} \approx 18.2\%$. Meanwhile, if the simulation is divided into 2 groups each of which has 10 nodes, the possibility that a group is delayed is $1 - 0.99^{10} \approx 9.6\%$. The need of synchronization across the groups depends on the data-dependencies among them. When needed, it is one-directional as

	TS	LCS
$T_A > T_B + \Delta t$	Stop A	-
$T_B + \Delta t \geq T_A > T_B - \Delta t$	-	-
$T_A \leq T_B - \Delta t$	Stop B	Stop B

Table 6.1: A comparison of necessary process stops for synchronizing nodes with different simulation times when node B has a data dependency on node A. T_X is the simulation time of node X.

in Equation 6.3. This allows some groups to run within a different simulation steps at a specific moment in the simulation. Therefore, the delays in some groups can be hidden by the delays in other groups at different simulation steps.

Second, Δt could be adjusted for the individual groups. This allows some groups that have less frequent communications to be synchronized with a longer simulation step. Compared to having a universal time step for the entire simulation, this largely reduces the overhead in synchronizing the groups where less frequent communications occur.

Table 6.1 compares TC and LCS with respect to when stopping a process is necessary in different situations. There are two nodes A and B and node B has a data dependency on node A. First, when the simulation time of node A exceeds the simulation time of node B by more than Δt, TC stops node A while LCS does not. LCS allows node A to be simulated faster than node B beyond Δt because node A has already generated the data that node B will access later.[3] Second, both methods do not stop any nodes when two nodes' simulation times are within Δt. Last, if the simulation of node A gets slower than node B, both methods stop node B. This stop cannot be avoided even in LCS, as the data that node A is supposed to generate is not yet ready when node B tries to access it. Hence, LCS stops node B until node A's simulation time catches up node B's simulation time.

[3]This is based on the assumption that the data accesses are non-blocking and one-directional. The data access in LCS is done through version-controlled files to handle modification or deletion of the data before the dependent node accesses it.

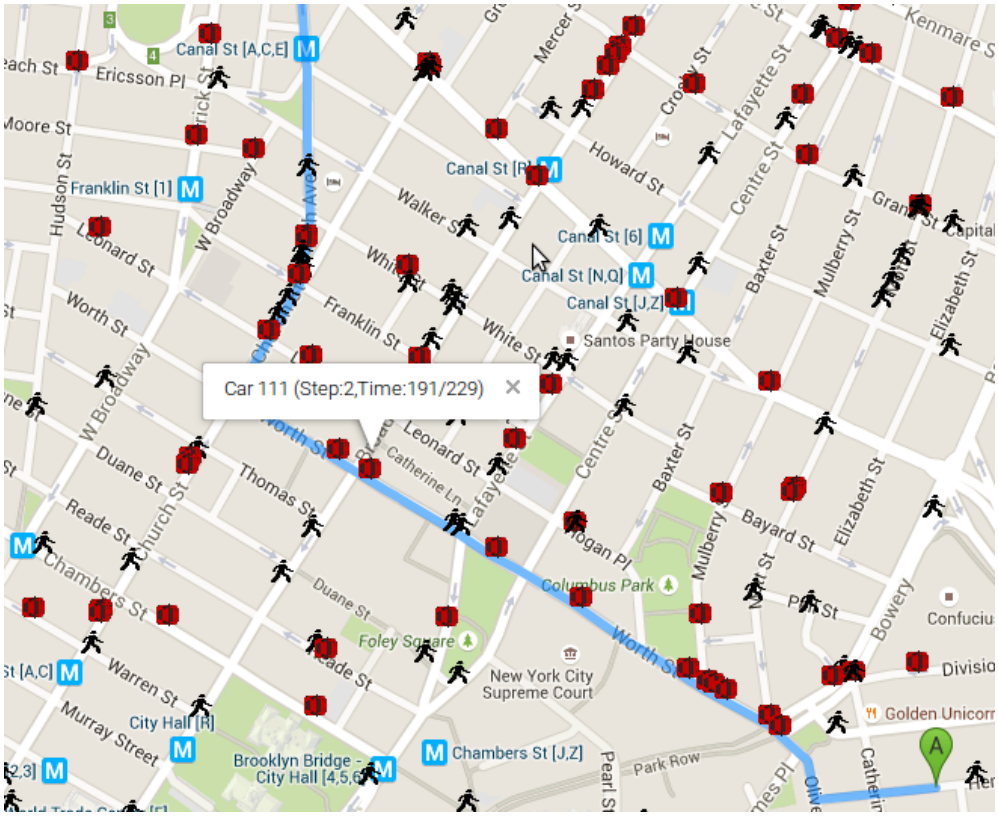


Figure 6.3: A web application for traffic simulation.

LCS is particularly efficient in the simulation of large-scale IoT systems where many sensor devices mostly generate data (as opposed to consume data), thus asynchronously affecting the rest of the system without the need to be synchronized with other sensors.

6.3 Case Study - A Traffic Management System

According to the World Resources Institute, the economic loss due to traffic congestion exceeds hundreds of millions US dollars annually in large cities, e.g., Seoul or Singapore [289]. In order to address these issues, various studies have been conducted in the emerging research fields of smart cities [33; 346] and advanced traffic management systems [153; 311]. A promising area is predicting and preventing traffic accidents [39]. However, accurate prediction is difficult because there are many variables that can affect the results, thus making the simulation complex. In this section, we apply SimbIoTics to the design of a TMS and perform various DSEs to find better design options across the complex design space.

Scenario. We developed an innovative TMS for a scenario based on the following assumptions:

- The target scope of this system is a metropolitan area where pedestrians and cars are in the street.
- There are some chances of an accident when at least two cars or a car and a pedestrian get close to each other.
- The probability of car accidents varies depending on the drivers (or the cars) and is higher in specific locations.
- The braking distance of vehicles is significantly affected by the temperature and the weather [92].

Figure 6.3 presents a map of a portion of Manhattan where a number of pedestrians and cars move toward their destination at their own speeds.[4] The case study is devised to reduce the possibility of traffic accidents and traffic jams by:

- alerting the pedestrians and vehicles;
- utilizing temperature sensors installed on the road to calculate braking distances for each car; and
- operating UAVs to monitor traffic and pedestrians.

System Implementation. The top half of Figure 6.4 shows the details of the TMS we designed. There are five distinct types of components in the system. The *traffic management server* controls and receives data from all other components in the system. It can move UAVs to other locations, change the length of traffic signals, and send a caution alarm to pedestrians. The *cameras* in traffic management are used for monitoring the traffic volumes and collect information on each vehicle. In the proposed system, a camera takes and analyzes images of passing-by vehicles to recognize their registration numbers. Then, it sends the numbers to the traffic management server. The *UAVs* monitor the vehicles and pedestrians in locations with a high risk of traffic accident using an accident-detection algorithm [151; 179] and report to the traffic management server if accidents occur. The *car history lookup service* gives the information on accident history of the requested

[4]We use the simulated traffic data created with a web application we developed. The application is available at *http://128.59.14.24/map*.

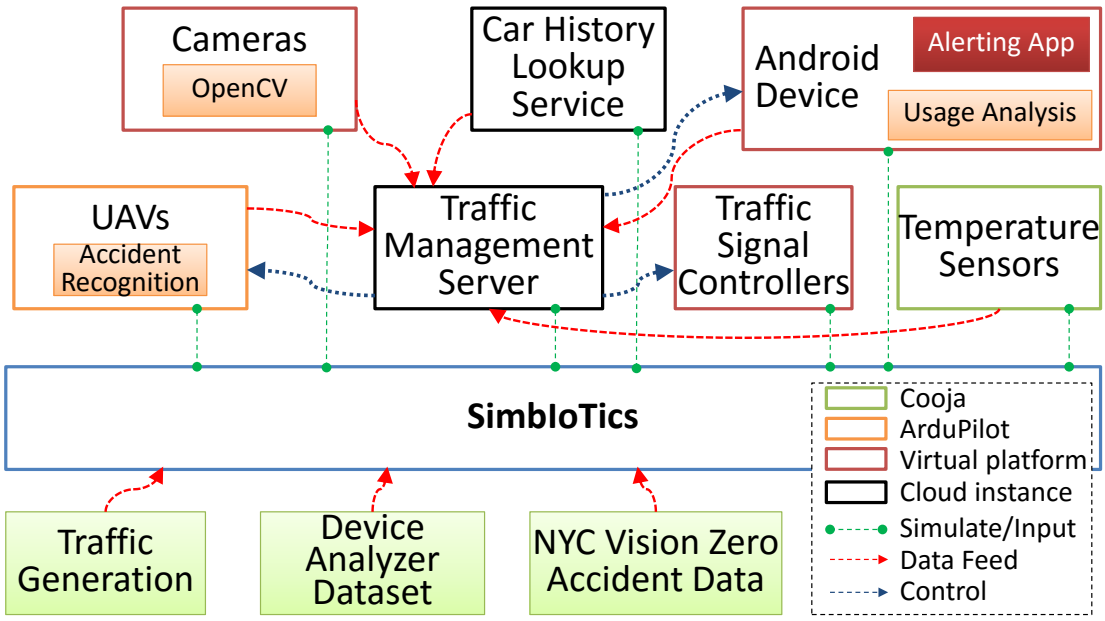


Figure 6.4: A traffic management system simulated on SimbIoTics.

vehicles. The *android devices* are smart devices carried by pedestrians: they report to the traffic management server where their users are and when they use them while walking; they also alert their users if the server requests them to do so. The TMS controls *traffic signals*, e.g., by lengthening a signal for slower pedestrians. It receives temperature data on the roads from the *temperature sensors*.

Simulating the System. We simulate the system by configuring SimbIoTics as shown in the bottom half of Figure 6.4. SimbIoTics orchestrates distinct simulated components that are synchronized as explained in Section 6.2.3. The Android devices, cameras, and traffic signals are simulated as embedded systems by using VPs. UAVs are run on ArduPilot. The network of sensors is simulated by Cooja.

In the data flow, the *input generation* module plays an important role. First, it exports the geolocation data of pedestrians and cars from the traffic generation web application in Figure 6.3. Instead of simulating directly many smart cars or pedestrians with smart phones, the input generation module feeds the data into the traffic management server. This separates traffic generation and management and allows the traffic data to be reused in different simulations, thus enabling a faster simulation speed and better scalability. Second, the input generation module uses the *Device Analyzer* dataset to simulate the behavior of the *usage analysis* module that catches when the

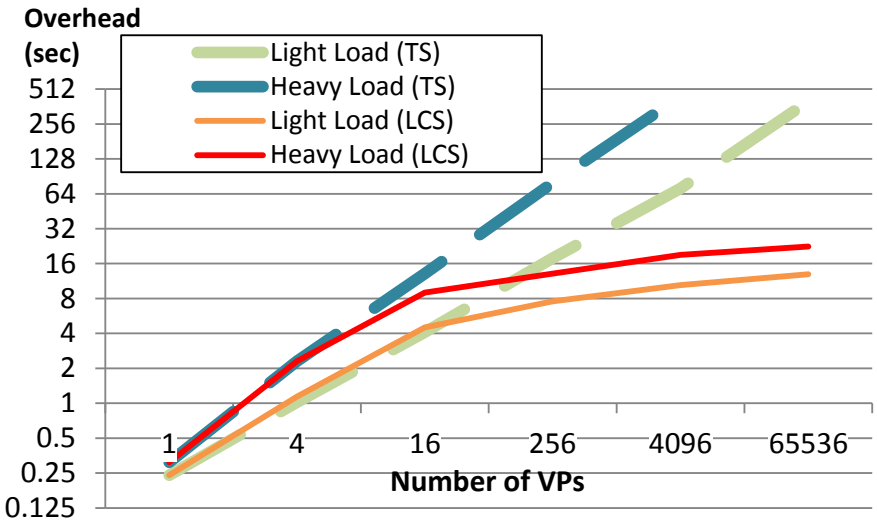


Figure 6.5: Performance comparisons between the traditional synchronization (TS) method and Loosely-Chasing Synchronization (LCS).

pedestrians use their phone while walking. The Device Analyzer dataset is data collected from over 23,000 Android devices, including when the users use their phones by calling, texting, or interacting with an app [325]. Third, we use accident data from the NYC Vision Zero project [17] to adjust the probability that a traffic accident occurs in a specific location. Additionally, SimbIoTics offers emulated input values to the simulated environments, e.g., it can generate random road temperature values for temperature sensors. When a new simulated vehicle is spawned, SimbIoTics randomly assigns an accident history to it in a way that can be used by various applications, e.g. to predict the probability of future accidents.

6.4 Experiments

In this section, we illustrate the capabilities of SimbIoTics by describing a set of experiments and design space explorations.

Synchronization. We compare the performance of LCS to the TS method introduced in Chapter 4. The TS method stops all the simulation processes whenever there are simulators whose simulation time is out of the allowed boundary (Δt) [167; 276]. Figure 6.5 shows the temporal overhead spent on synchronizing the VP nodes. The overhead is measured as the sum of the time periods when all VPs in the simulation have to stop for synchronized simulation. The nodes in the

		4096		65536	
		Light	Heavy	Light	Heavy
TS		184.7	2358.2	1241.4	4485.5
LCS	in-group	15.3	32.2	24.5	38.1
	across-group	5.7	12.9	10.4	17.7
	Sum	21.0	45.1	34.9	55.8

Table 6.2: Number of stopped process during synchronization.

LCS are grouped to form a graph as discussed in Section 6.2.3 with a maximum group size of 16. We prepare two distinct types of loads that we input to the VPs to simulate in order to measure the synchronization overhead. A heavy load involves simulation nodes executing repeatedly the case study application described in Section 6.3. A light load includes approximately 80% of idle time. The experimental results show that the increase rate of LCS becomes notably low as the number of VPs in the simulation increases. With 4096 VPs, the overhead is more than an order of magnitude less on LCS than TS. This confirms that LCS is much more scalable than TC, which synchronizes all the simulated nodes at the same time, particularly when it comes to large-scale environments like IoT systems. Table 6.2 presents the counts of how many times the synchronizer stopped a process, which is the main cause of the overhead when simulating 4096 and 65536 nodes. The number of stopped process is divided in two sets: 1) when the dependent process and the independent processes are within the same group (in-group) and 2) when the two processes belong to two different groups (across-group). Although the number of stops is not directly proportional to the overhead (as the duration of each stop differs from one to another), the overall overhead growth tends to follow the number of simulation process stops. Table 6.2 shows that the performance gain in LCS is achieved for both in-group and across-group cases. The result also indicates that the performance gap between TC and LCS is higher under heavy load.

Performance Evaluation on Large-Scale, Heterogeneous Simulation. Figure 6.6 presents the time spent on simulating 4096 to 262,144 nodes. Each simulation involves a heterogeneous set of nodes including: cloud instances (0.4%), embedded systems (6%), UAVs (3%), and sensor devices (90.6%) for 1,000 seconds simulation time. As we increase the number of VMs used for the simulation, the overall execution time increases very slowly after reaching about 2500 seconds.

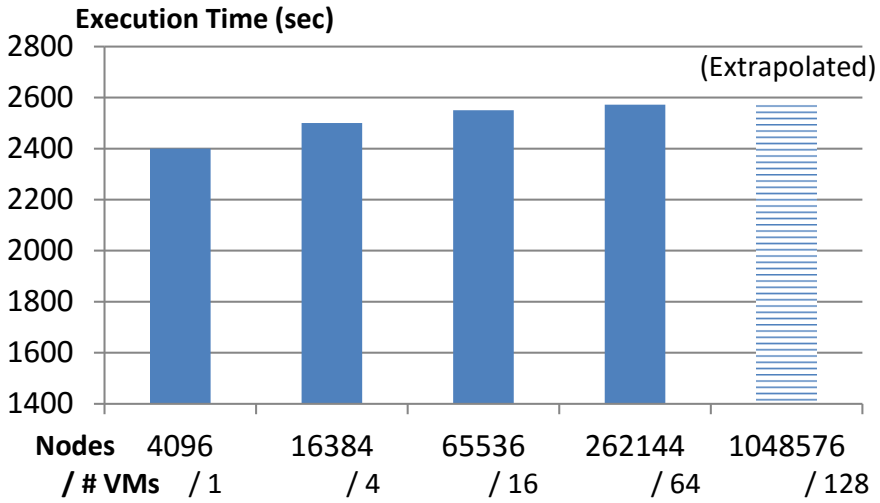


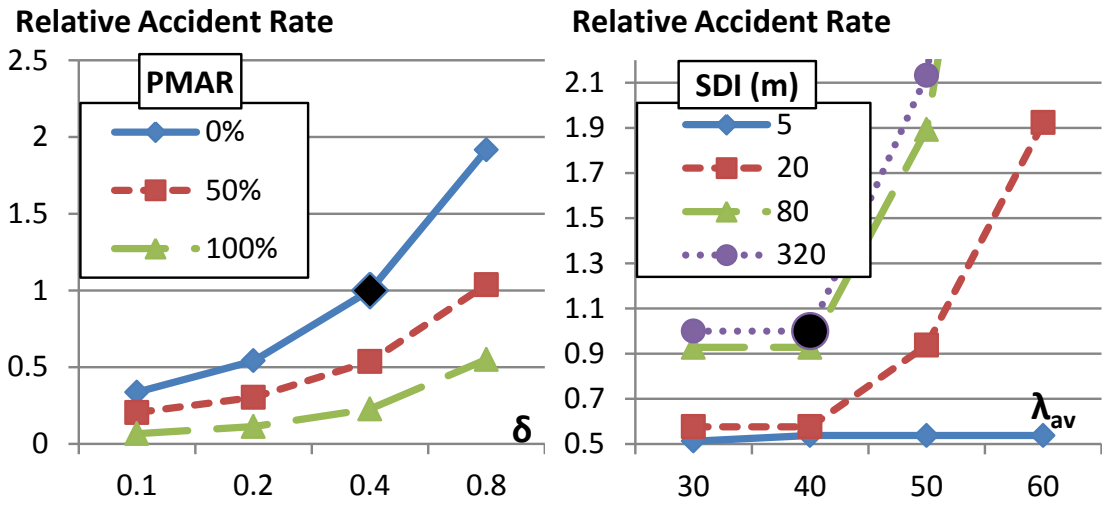
Figure 6.6: Execution time of large-scale simulation.

	Area A			Area B		
Traffic	25%	50%	100%	25%	50%	100%
Paramics	0.752	0.660	0.548	0.793	0.789	0.682
VISSIM	0.722	0.734	0.719	0.794	0.788	0.761
Interpol.	0.791	0.546	0.370	0.787	0.635	0.477
Random	0.321	0.291	0.273	0.418	0.327	0.360

Table 6.3: Prediction accuracy of four vehicle models.

Extrapolating, we expect that the execution time for simulating a million nodes for 1000 seconds still stays under 2600 seconds, yielding 38.6% as a simulation-realtime ratio. This suggests that SimbIoTics could scale efficiently to support the simulation of over a million nodes given a sufficient number of host VM instances.

Vehicle Prediction Model Evaluation. One of the functions SimbIoTics can perform is evaluating behavioral models of the simulated components. Due to the limited number of installed sensors and limited network availability, a TMS might collect only a fraction of the information on vehicle geolocation over time. However, many features of the TMS, such as accident prediction or traffic signal management, rely on the modeling and prediction of this information. SimbIoTics can be used to evaluate existing vehicle models [59; 213] and find the one that works most accurately for the target application. Table 6.3 compares the accuracy of four vehicle models when the



(a) NAR as a function of PMAR and δ (Baseline: PMAR=0%, δ=0.2)

(b) NAR as a function of SDI and λ (Baseline: SDI=320m, λ=4.8m)

Figure 6.7: Normalized Accident Rates (NAR).

traffic volume changes (25% ~ 100% of the given base volume). This experiment is based on the Mobile Century traffic data [139]. Paramics proposes a queue-based vehicle model, called *Car Following* [59]. VISSIM offers a vehicle model using driver behavior parameters such as the distance between cars and vehicle speed preferred by drivers [213]. We added two simple models: Interpolation, which prorates the distance and the time across the shortest path between two measured spots, and Random, which assumes the vehicle has gone through a random path reachable with a given maximum speed, e.g., 70mph. The accuracies are measured as the ratio of incorrectly predicted location for each time unit.[5] By using SimbIoTics we can determine that VISSIM works best in most cases while the Interpolation model works well when the traffic volume is low.

Large-Scale Accident Prediction and Prevention. Thanks to SimbIoTics scalability, we can perform accident prediction based on a large-scale simulation. For instance, simulating a metropolitan area like New York City requires to account for millions of vehicles and pedestrians that incur hundreds of thousands accidents each year. SimbIoTics can help engineers to perform various large-scale analyses. For example, we can use it to evaluate the effectiveness of connecting the vehicles to the Internet in order to reduce the accident rates. Suppose that the probability P that an accident

[5]The correctness of location prediction is determined by a given range, e.g., 0.01 mile.

happens between two vehicles, v_i and v_j is:

$$P(v_i, v_j) = min(\frac{R}{2^{d(v_i,v_i)}}, \delta) \tag{6.4}$$

where $0 < R < 1$ is a uniformly distributed random number and $d(v_i, v_j)$ is the distance between v_i and v_j in meters. δ is an adjustable user parameter which captures the maximum accident probability, e.g., 0.2. Another important metric is the Prevention Method Adoption Ratio (PMAR) that counts how many vehicles have adopted the alert-based prevention method. Figure 6.7(a) shows the accident occurrence normalized to the baseline (0% PMAR and 0.2 Δ). Through this experiment using SimbIoTics, we can anticipate that the adoption of the accident prevention method can reduce the vehicle accidents by approximately 60% to 80%. In addition, SimbIoTics can help the users to build the accident probability model, for example, by adjusting the value of δ.

DSE for Cost Efficiency. The braking distance[6] is critical in accident prevention and is largely affected by the road surface temperature [92]. We can use SimbIoTics for DSE on the number of installed sensors and the accident rate. The installation of temperature sensors on the road as part of an augmented TMS could reduce the vehicle accident rate by providing a new tool to control the speed of vehicles on the area where the braking distance is longer than usual. By performing a DSE analysis with SimbIoTics, engineers can estimate to which extent the number of sensors that could be installed on the road could reduce the accident rate. We assume a simplified accident probability P' of a vehicle v_i as:

$$P'(v_i) = 0.01R \cdot max(1, 2^{\lambda-\lambda_{av}}) \cdot \Big(1 - min(1, \frac{SCR}{SDI})\Big) + \sum_n P(v_i, v_n) \tag{6.5}$$

where λ is the braking distance that the driver usually anticipate, e.g., 40 meters, λ_{av} is the average of actual braking distances, Sensor Coverage Range (SCR) is 15 meters, and Sensor Distance Interval (SDI) denotes how far from one temperature sensor to another. Thus, as the braking distance increases beyond the anticipated distance, the accident rate increases exponentially. Figure 6.7(b) presents the normalized accident rates obtained on SimbIoTics. As expected, the results show that a smaller value of SDI reduces the accident rates, which, however, incurs high costs for sensor purchase, installation, and maintenance. Also, through this kind of DSE, the users of SimbIoTics can achieve insights on various design trade-offs. For instance, when λ is 40 meters, installing and

[6]The distance that a vehicle will travel after its brakes are applied.

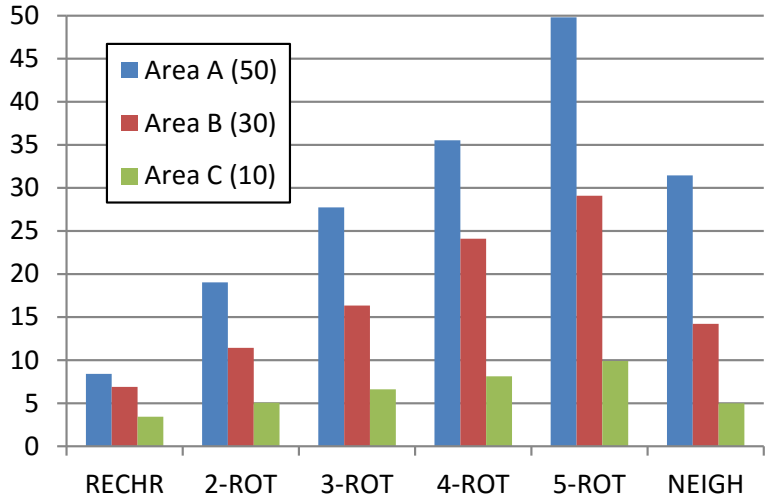


Figure 6.8: Coverage vs. recharging methods.

managing sensor devices at an SDI of 5 meters as opposed to 20 meters leads to a slight reduction of accident rate from 45% to 43%.

UAV Deployment. Recent traffic-signal systems adjust the duration of each signal according to the duration of the time vehicles and pedestrians have been waiting. Therefore, the presence of many pedestrians in a given area is more likely to cause a traffic jam. To avoid routing vehicles to areas with many pedestrians, some TMSs use UAVs. We simulate a TMS where UAVs monitor the pedestrians to predict the areas where they will likely gather and to route vehicles through less condensed paths. An important constraint of current UAVs is that they can fly continuously for only about half an hour without recharging its battery. There are multiple options to handle the recharging issue of UAVs: (1) stop monitoring for UAVs while recharging (RECHR), (2) assign a group of N UAVs to one area where only one drone monitors while others recharge (N-ROT)[7], or (3) recharge the UAVs in need and expand nearby UAVs' to cover the monitoring of those areas originally assigned to the recharging one (NEIGH). Figure 6.8 shows how many condensed pedestrian groups (CPGs) the UAVs detected from areas with different CPGs across different recharging methods. For instance, the UAVs detected five CPGs out of ten in Area C. This kind of analysis allows us to determine which recharging method is best for a given environment by considering the costs for number of UAVs and the desired level of coverage of the CPGs.

[7]RECHR is a special case of N-ROT, i.e., 1-ROT.

6.5 Related Work

Most of the existing simulation frameworks for IoT systems are focused on supporting only the simulation of sensor devices [94; 200]. These frameworks do not capture one important characteristic of IoT systems: the integration of sensors and cloud computers. This is a critical aspect because it enables the aggregation and analysis of the huge amount of information collected from a myriad of sensor devices.

The performance of synchronization mechanisms in the simulation of distributed embedded systems has been studied before [82; 167], but the scale of the systems simulated in these projects was limited to hundreds to tens of thousands nodes.

6.6 Concluding Remarks

We have developed SimbIoTics, an innovative simulation framework to support the design and development of many classes of IoT systems. To provide the capabilities of modeling many heterogeneous components we integrated within SimbIoTics multiple cloud instances together with various virtual platforms and simulators for wireless sensor networks and UAVs. To increase the scalability of SimbIoTics, we developed a synchronization method that leverages the dependencies among the nodes in the system. By presenting a series of analysis for a traffic management system, we illustrated the capabilities of SimbIoTics to support modeling and design-space exploration for IoT.

Part II

A Spectrum of Distributions of Computations in MCC Systems

Chapter 7

Reverse Distributed Offloading: a Cluster of Embedded Systems

An expanding wealth of ubiquitous, heterogeneous, and interconnected embedded devices is behind most of the exponential growth of the "Big Data" phenomenon. Meanwhile, the same embedded devices continue to improve in terms of computational capabilities, thus closing the gap with more traditional computers. Motivated by these trends, together with Richard Neill and Luca Carloni, I developed a heterogeneous computing system for MapReduce applications that couples cloud computing with distributed embedded computing. Specifically, our system combines a central cluster of Linux servers with a broadband network of embedded set-top box (STB) devices. The MapReduce platform is based on the Hadoop software framework, which we modified and optimized for execution on the STBs. Experimental results confirm that this type of heterogeneous computing system can offer a scalable and energy-efficient platform for the processing of large-scale data-intensive applications. This project has led to the need for design tools for large-scale, heterogeneous systems and has been the initial motivation of developing NETSHIP.

7.1 Introduction

The growth in the amount of data created, distributed and consumed continues to expand at exponential rates: according to a recent research report from the International Data Corporation, the amount of digital information created and replicated has exceeded the zettabyte barrier in 2010 and

this trend is expected to continue to grow "as more and more embedded systems pump their bits into the digital cosmos" [148]. In recent years the MapReduce framework has emerged as one of the most widely used parallel computing platforms for processing data at very large scales [174]. While MapReduce was originally developed at Google [85], open-source implementations such as Hadoop [3] are now gaining widespread acceptance.

The ability to manage and process data-intensive applications using MapReduce systems such as Hadoop has spurred research in server technologies and new forms of Cloud services such as those available from Yahoo, Google, and Amazon.

Meanwhile, the Information Technology industry is experiencing two major trends. On one hand, computation is moving away from traditional desktop and department-level computer centers towards an infrastructural core that consists of many large and distributed data centers with high-performance computer servers and data storage devices, virtualized and available as Cloud services. These large-scale centers provide all sorts of computational services to a multiplicity of peripheral clients, through various interconnection networks. On the other hand, the increasing majority of these clients consist of a growing variety of embedded devices, such as smart phones, tablet computers and television set-top boxes (STB), whose capabilities continue to improve while also providing data locality associated to data-intensive application processing of interest [234; 233]. Indeed, the massive scale of today's data creation explosion is closely aligned to the distributed computational resources of the expanding universe of distributed embedded systems and devices. Multiple Service Operators (MSOs), such as cable providers, are an example of companies that drive both the rapid growth and evolution of large-scale computational systems, consumer and business data, as well as the deployment of an increasing number of increasingly powerful embedded processors.

The contributions of this chapter are motivated precisely by the idea that the ubiquitous adoption of embedded devices by consumers and the combination of the technology trends in embedded systems, data centers, and broadband networks open the way to a new class of heterogeneous Cloud computing for processing data-intensive applications. In particular, we propose a *broadband embedded computing system for MapReduce utilizing Hadoop* as an example of such systems. Its potential application domains include: ubiquitous social networking computing, large-scale data mining and analytics, and even some types of high-performance computing for scientific data analy-

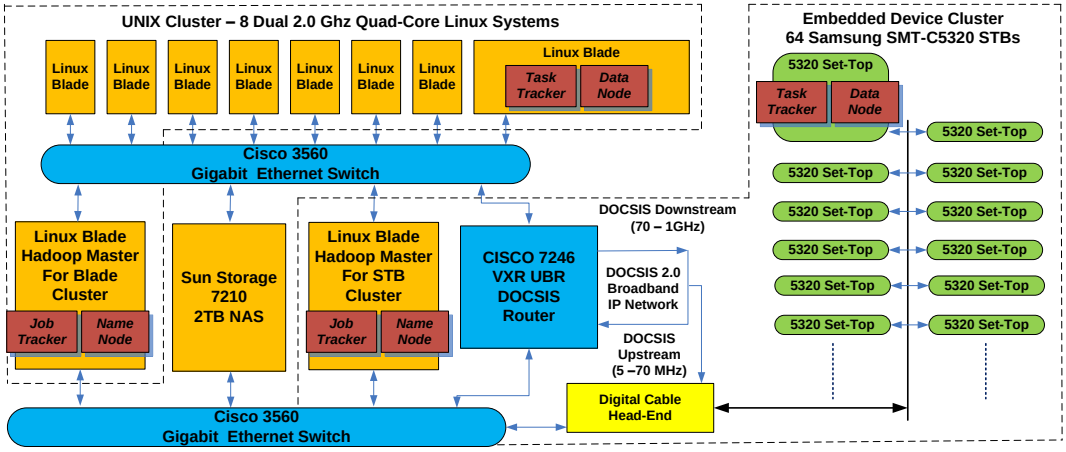


Figure 7.1: Architecture of the broadband embedded computing system for MapReduce utilizing Hadoop.

sis. We present a heterogeneous distributed system architecture which combines a traditional cluster of Linux blade servers with a cluster of embedded processors interconnected through a broadband network to offer massive MapReduce data-intensive processing potential (and, potentially, energy and cost efficiency).

Contributions. We have implemented a prototype small-scale version of our proposed system where a *Linux Cluster* features nine high-end blade servers and an *Embedded Cluster* consists of a network of 64 STBs. The two clusters are interconnected through the broadband network of a complete head-end cable system, as described in Section 7.2. While the cable system remains fully operational in terms of its original function (e.g., by distributing streaming-video content to the STBs which render it to their displays), it is possible to simultaneously and effectively execute other MapReduce applications by leveraging the additional computation resources that are available in the STB multi-core processors.

Specifically, we ported the Hadoop MapReduce framework to our broadband embedded computing system. As discussed in Section 7.3, this porting posed important challenges in terms of software portability and resource management. We addressed these challenges in two ways. First, we developed porting techniques for embedded devices that leverages back-porting of enterprise software in order to implement the Hadoop system for embedded environments. Second, to execute MapReduce applications on such resource-constrained embedded devices as STBs, we optimized both memory and storage requirements by eliminating unnecessary software components of the

Hadoop platform. The result is an embedded version of the Hadoop framework.

In Section 7.4 we present a set of experiments which confirm that our embedded system implementation of the Hadoop runtime environment and related software libraries runs successfully a variety of MapReduce benchmark applications. Also, in order to gain further insight into the relative performance scaling of the Embedded Cluster versus the Linux Cluster while running MapReduce applications, we varied the number of processing elements (which correspond to the number of *Hadoop nodes*) and the size of the input data. Overall, the experimental results expose the Embedded Cluster performance sensitivity to certain classes of MapReduce applications and indicate avenues of future research to improve our system.

7.2 The System Architecture

Figure 7.1 provides an overview of the architecture of the system that we developed and built: this is a heterogeneous system that leverages a broadband network of embedded devices to execute MapReduce applications by utilizing Hadoop. It is composed of four main subsystems.

Linux Blade Cluster. The Linux Cluster consists of a traditional network of nine blade servers and a Network Attached Storage (NAS). Each blade has two quad-core 2GHz Xeon processors running Debian Linux with 32GB of memory and a 1Gb/s Ethernet interface. One of the nine blades is the Hadoop master host acting both as NameNode and JobTracker for the MapReduce runtime management [3]. Each of the other eight blades is a Hadoop slave node, acting both as DataNode and TaskTracker [3] while leveraging the combined computational power of the eight processing cores integrated on the blade. The blades use the Network File System (NFS) to mount the 2TB Sun storage array which provides a remote common file-system partition to store applications for each of the executing Hadoop MapReduce applications. For storing the Hadoop Distributed File System (HDFS) data, the blades use their own local hard-disk drive (HDD).

Embedded STB Cluster. The Embedded Cluster consists of 64 Samsung SMT-C5320 set-top boxes (STB) that are connected with a radiofrequency (RF) network for data delivery using MPEG and DOCSIS transport mechanisms. The Samsung SMT-C5320 is an advanced (2010-generation) STB featuring an SoC with a Broadcom MIPS 4000 class processor, a floating-point unit, dedicated video and 2D/3D-graphics processors with OpenGL support, 256MB of system memory, 64MB in-

ternal Flash memory, 32GB of external Flash memory accessible through USB, and many network transport interfaces (DOCSIS 2.0, MPEG-2/4 and Ethernet). Indeed, an important architectural feature of modern STBs is the heterogeneous multi-core architecture design which allows the 400MHz MIPS processor, graphics/video processors, and network processors to operate in parallel over independent buses. Hence, user-interface applications (such as the electronic programming guides) can execute in parallel with any real-time video processing. From the viewpoint of running Hadoop applications as a slave node, however, each STB can leverage only the MIPS processor while acting both as DataNode and TaskTracker. [1] This is an important difference between the Embedded Cluster and the Linux Cluster. Finally, in each STB, a 32GB USB memory stick is used for HDFS data storage, while NFS is used for Java class storage.

Digital Cable Head-End. This is responsible for controlling the Embedded Cluster devices and providing all interactive television services including: electronic program guide, user-interface, video-on-demand (VOD), and the delivery of MPEG-2 videos. Our digital head-end supports the current generation of STBs based on the Cablelabs Tru2way standard [27] and is a scaled-down but complete implementation of a modern digital DOCSIS-based broadband cable system in-use at today's largest MSOs. As shown in Figure 7.1, its core components include: 1) the Tru2way Object Carousel for MPEG-2 delivery of Embedded Cluster applications and Tru2way-standard STB signaling; 2) two Linux hosts for TCP/IP DHCP and TFTP network services, which are required for assigning system-wide IP addresses and DOCSIS cable-modem configuration data to all Embedded Cluster devices; 3) an HTTP application/data server that supports interactive television services via TCP/IP over the DOCSIS network; 4) support for MPEG-2 video sources that are multiplexed and grouped into digital channels, including a single channel for VOD streams; and 5) a RF distribution and combining network that utilizes a Cisco QAM modulator device to translate digital input signals from the carousel, multiplexed MPEG sources, and VOD server, into modulated QAM256 RF frequencies, which can be combined with the DOCSIS router RF output to feed the broadband network of STBs.

DOCSIS is a standard broadband-network technology for TCP/IP over RF cable. It provides for an inter-operable RF modem, based on TDMA protocols organized in a star topology connecting the

[1] In the Embedded Cluster, there is also a Linux blade which is the Hadoop master node, acting both as NameNode and JobTracker.

central router and the STBs. The SMT-C5320 DOCSIS 2.0 TCP/IP and MPEG-2 transport stream interfaces use quadrature amplitude modulation (QAM) protocols for transmitting and receiving signals on North American digital cable systems. Devices on DOCSIS share access to the network, as arbitrated by the central router, and operate effectively at up to 27Mbps in the downstream direction (towards the STB) and 27Mbps in the upstream direction (towards the cluster). The MPEG-2 interface is primarily used for decoding video programs, but can also receive applications or data delivered via the Tru2way Object Carousel (OC), a "broadcast file system" service on a dedicated QAM frequency. This data is sent from the head-end at regular intervals over MPEG-2 directly into a QAM device where it is modulated onto the RF cable plant at a specified frequency for STB reception. Broadcast applications are STB executables or data that are simultaneously available to all STBs connected to the broadband network. A STB device tunes to a specific channel frequency and receives the application/data of interest according to the Tru2way protocol. The carousel may also deliver Tru2way signaling and other forms of data over DOCSIS as multicast group messages following the DOCSIS Set-top Gateway, or DSG protocol [8]. In our prototype system this data-delivery mechanism is used to control the STB boot-up and user-interface applications.

Network. The system network is a managed dedicated broadband network which is divided into three IP subnets to isolate the traffic between the DOCSIS-based broadband Embedded Cluster network, the Linux Cluster network, and the digital cable head-end. Its implementation is based on two Cisco 3560 1Gb/s Ethernet switches and one Cisco 7246 DOCSIS broadband router. The upper switch in Figure 7.1 interconnects the eight blades along with the NAS and master host. The lower switch aggregates all the components on the head-end subnetwork. The DOCSIS subnetwork is utilized by the Embedded Cluster whose traffic exists on both the Linux Cluster and the digital head-end network. The broadband router has 1Gb/s interfaces for interconnection to the Linux Cluster and head-end networks as well as a broadband interface for converting between the DOCSIS network and the Ethernet backbone. Each broadband router can support over 16,000 STBs, thus providing large-scale fan-out from the Linux Cluster to the Embedded Cluster.

Embedded Middleware Stack. The embedded middleware stack is based on Tru2way, a standard platform deployed by major cable operators in U.S. as part of the Open Cable Application Platform (OCAP) developed in conjunction with Cablelabs [8]. Various services are delivered through the Tru2way platform including: chat, e-mails, electronic games, video on-demand (VOD), home

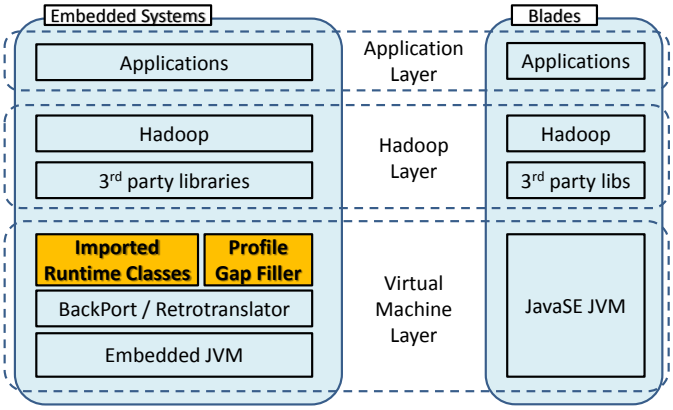


Figure 7.2: Two software stacks to support Hadoop: STB vs. Linux Blade.

shopping, interactive program guides, stock tickers, and, most importantly, web browsing [27]. To enable cable operators and other third-party developers to provide portable services, Tru2way includes middleware based on Java technology that is integrated into digital video recorders, STBs, TVs, and other media-related devices.

Tru2way is based on Java ME (Java Micro Edition) with CDC (Connected Device Configuration) designed for mobile and other embedded devices. The Tru2way standard follows FP (Foundation Profile) and PBP (Personal Basis Profile) including: io, lang, net, security, text, and util packages as well as awt, beans, and rmi packages, respectively. Additional packages include JavaTV for Xlet applications, JMF (Java Media Framework), which adds audio, video, and other time-based media functionalities, and MHP (Multimedia Home Platform), which comprises classes for interactive digital television applications. On top of these profiles, the OCAP API provides applications with Tru2way-specific classes related to hardware, media, and user-interface packages unique to cable-based broadband content-delivery systems.

Remark. While this rich set of Java profiles offer additional features to the embedded Java applications, there exists a significant gap between the Java stack provided by Tru2way and the Java Platform Standard Edition (Java SE), which is common to enterprise-class application development. Hence, since the standard Hadoop execution depends on the Java SE environment, we had to develop a new implementation of Hadoop specialized for the embedded software environment that characterizes devices such as STBs. We describe our effort in the next section.

7.3 Porting Hadoop to the Cluster of Embedded Systems

There are several issues that need to be addressed in order to successfully run Hadoop on a distributed embedded systems like our broadband network of STB devices.

First, Hadoop and Hadoop third-party libraries require many bootstrap classes not supported by the Tru2way JVM. Also, for many classes the Tru2way JVM supports only a subset of methods: e.g., both Tru2way and Java SE have the `java.lang.System` class, but the `java.lang.System.getenv()` method exists only in Java SE.

Second, the Tru2way JVM only supports older versions of Java class file formats while Hadoop is developed using many Java 1.6 language features including: generics, enums, for-each loops, annotations, and variable arguments.

Third, the task of porting Java applications to another JVM with different profiles is quite challenging and, differently from porting native codes to JVM [43; 170], it has not been actively studied in the literature. If not an impossible task, to modify Hadoop and the Hadoop third-party libraries at the source code level is not really practical because there are more than fifty of such libraries and, in some cases, their source code is not available.

Finally, despite all the efforts to improve the JVM portability [250; 278], to port the Java SE JVM to the STB environment is difficult because these embedded devices do not support key features such as frame buffer or native implementations.

To address these challenges, we have developed a binary-level porting method for embedded devices that imports missing class files and retro-translates all the class files so that the embedded Tru2way JVM can execute them. Our method leverages the Java Backport package, which is the implementation of JSR 166 (`java.util.concurrent` APIs), introduced in Java SE 5.0 and further refined in Java SE 6.0, for older versions of Java platforms [7]. The Retrotranslator has two main functionalities: 1) it translates newer class files into an older format for an older JVM; and, 2) it extends the Backport package so that most Java SE 5.0 features are available for an application that runs on the Java SE 1.4 and Java SE 1.3 JVMs [23]. The runtime classes from those two packages can be added to the Tru2way JVM.

Figure 7.2 shows the resulting software stack to support the execution of Hadoop in the embedded environment of an STB running the Tru2way JVM and contrasts it with the traditional software stack based on the Java SE JVM running on a common Linux blade. In particular, the embedded

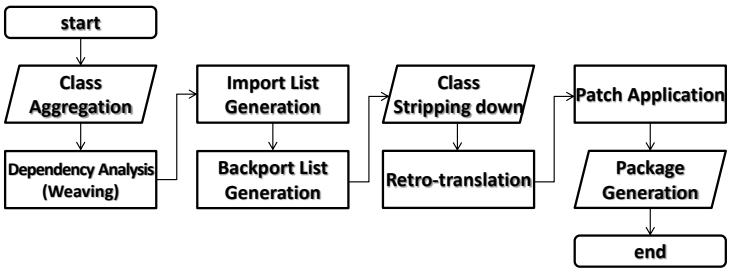


Figure 7.3: Porting the Hadoop-supporting Java classes to the STB devices.

software stack includes the *Imported Runtime Classes*, which are the results of the backporting technique, and the *Profile Gap Filler*, which collects all additional components that were developed specifically for the embedded STB devices.

Figure 7.3 illustrates the procedure that we developed to port Hadoop and all the Java packages necessary for running Hadoop to the STB devices. While it was developed and tested for our broadband embedded system, for the most part this procedure is a contribution of general applicability to port Java applications originally developed for the Java SE JVM to other embedded systems which have different and more limited JVMs: e.g., this procedure can be followed also for porting any Java applications to other JVM such as BD-J or Android's Dalvik [128; 220]. The procedure consists of a sequence of eight main steps:

1) Class Aggregation. Here all the input classes are simply copied into a single directory and the priorities among the duplicated or collided classes are determined.

2) Dependency Analysis. For this step, which is key to implementing efficiently a large Java application like Hadoop on resource-constrained embedded devices, we developed a novel dependency analysis technique called *Class Weaving*. This starts by analyzing the class dependencies within a Java package as well as across the packages and then changes the dependency to reuse as much as possible those classes which are available in the embedded Java ME environment. The goal is to generate all the information on class dependencies that is necessary at later steps to minimize the number of classes which will be imported from the various open-source Java SE runtime libraries (and to strip out all unnecessary classes from the original packages.) Figure 7.4 illustrates how Class Weaving works: a class dependency tree is generated by analyzing each class while minimizing the number of classes to be imported. For example, Hadoop's `TaskTracker` class uses the `Pattern` class, which in turn uses the `Matcher` class: both these classes exist in Java SE but not in the STB Java ME environment and, therefore, need to be imported. On the other hand, the

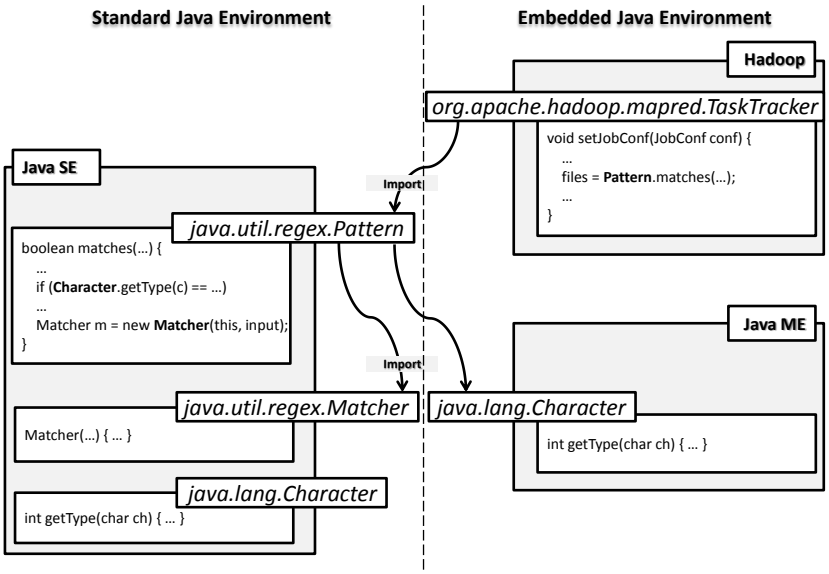


Figure 7.4: Example of applying the proposed Class Weaving method.

`Pattern` class uses the `Character` class, which exists also in Java ME and, therefore, it will not be imported from Java SE: instead, the `Pattern` class will be woven to use Java ME's `Character` class.

3) Import List Generation. Based on the information collected at the previous step, the list of classes to be imported is generated. At this step, the list can be refined through additional customizations. Unlike most JVMs, some embedded JVMs have their bootstrap classes embedded in a way that are not accessible to the application developers and provide only stub classes to them. For instance, packages like `xerces` or `log4j` do exist in the actual bootstrap classes for internal purposes but are not included in the stub classes.

4) Backport List Generation. The Java class loaders check if the package name of the target class begins with the `'java.'` prefix when the class file location is not in the bootstrap classpaths and, if so, returns an error. To avoid this, the prefix needs to be changed: e.g., in the case of our system with the `'edu.columbia.cs.sld.backport.ocap.java.'` prefix. A list of the mappings between the original and the new prefix is generated for all the imported classes with package names that begin with `'java.'` to be used later in the retro-translation step.

5) Class Stripping Optimization. Since many embedded systems have limited memory and storage resources, only the necessary Java classes should be stored in the embedded device. This is achieved by collecting dependency trees that begin with the *seed classes*, which include the entry

	Hadoop & 3rd-party libs	JavaSE Bootstrap
Before	14490	10110
After	4141	5978

Table 7.1: Class count before & after class stripping optimization.

point `Xlet` class that launches Hadoop DataNode and TaskTracker, various classes that are dynamically loaded from configuration files or from the source code, and the patched classes. In our case, this step results in a 60% reduction of the number of classes that must be deployed in the STBs, as shown in Table 7.1.

6) Retro-translation. Since the Tru2way JVM recognizes classes up to Major Version Number 48, all the class files with Major Version Number 49 or higher need to be retro-translated. Most packages, including Hadoop, provide classes with major version number 50, which corresponds to Java 1.6. At the binary level, the class file formats and package names of Hadoop, Java SE, and the application libraries need to be properly modified.

7) Patch Application. While a number of classes were imported from open-source Java SE runtime libraries through the Class Weaving technique described above, we had to newly develop a number of missing classes and methods which needed to be optimized before being added to the Java stack of the STBs. The same was necessary for classes that could not be imported from the open-source Java SE runtime library due to the native implementations. Also, patches were necessary to fix some defects found in the Tru2way implementations.

8) Package Generation. This final step generates the packages that will be launched on the Tru2way JVM from the stripped classes, links a custom class loader that will loads user-defined `Mapper` and `Reducer` classes, and binds an entry point `Xlet` that will execute Hadoop DataNode and TaskTracker.

7.3.1 Challenges in Porting Hadoop to STB Devices

The number of JVM processes supported in the system is one of the biggest differences between the STB Java environment and a Linux blade server utilizing Java SE. While the users of the latter can launch multiple instances of JVM, only one JVM instance can be launched during boot time

within an STB. On the other hand, there are two important behaviors in Hadoop that rely on the capability of multiple JVM executions: first, TaskTracker and DataNode are running two different JVM processes; second, for each task processed in a TaskTracker node, a new JVM instance is launched unless there is an idle JVM which can be reused for the task.

To support these behaviors while coping with the STB limitation of running only one JVM instance, we implemented a new `ProcessBuilder` class that creates a thread group whenever the launch of a new JVM process is requested. Each thread group provides a distinct set of Hadoop environmental variables which are managed within the threads belonging to a given thread group without interfering with other threads groups. The `ProcessBuilder` class implementation also enables optimizations such as replacing IPC (Inter-Process Call) with method invocations in the same process, and the elimination of local data transfers through sockets with local file-copy operations.

A number of other middleware issues related to porting Hadoop to an embedded device like the STB were discovered and resolved. For example, certain Java classes have bugs that make the application behave improperly, halt, or sometimes fail. In these cases the classes were replaced with better implementations or patched to align with the Hadoop Java class requirements. Also, some configuration changes were made to the system: e.g., the Socket timeout constant had to be slightly extended to account for variations in network response times or delays. Finally, to relieve memory constraints, we reduced the number of threads associated to unimportant services such as the metrics service which profiles the statistics of performance or the web service that provides status information.

7.4 Experiments

In order to evaluate our embedded Hadoop system for its scalability characteristics and execution performance, we executed a number of MapReduce experimental tests across the Linux Cluster and Embedded Cluster. All the experiments were performed while varying the degree of parallelism, i.e., by iteratively doubling the number of Hadoop nodes, of each cluster: specifically, from 1 to 8 Linux blades for the Linux Cluster (where each blade contains eight 2GHz processor cores) and from 8 through 64 STBs for the Embedded Cluster (where each STB contains one 400MHz processor core). The results can be organized in four groups which are presented in the following

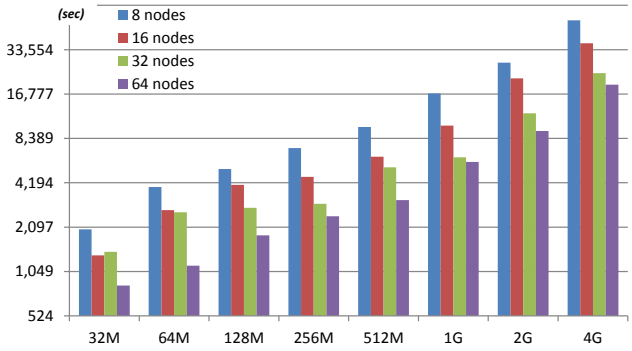


Figure 7.5: *WordCount* execution time as function of problem size (bytes), node count on Embedded Cluster (each node is one STB).

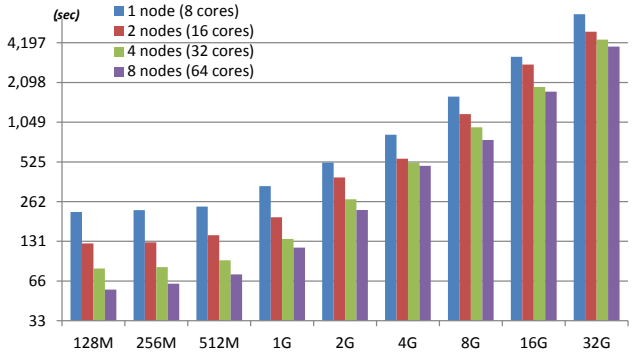


Figure 7.6: *WordCount* execution time as function of problem size (bytes), node count on Linux Cluster (each node is a 8-core blade).

subsection. We report the average results after executing all tests multiple times.

7.4.1 The WordCount Application

WordCount is a typical MapReduce application that counts the occurrences of each word in a large collection of documents. The results reported in Figure 7.5 and 7.6 show that this application scales consistently for both the Embedded Cluster and Linux Cluster. As the size of the input data increases, the Embedded Cluster clearly benefits from the availability of a larger number of STB nodes to process larger data sets. The Linux Cluster execution time remains approximately constant for data sizes growing from 128MB to 512MB since these are relatively small, but then it begins to double as the data sizes grow from 1GB to 32GB. In fact, above the 1GB threshold the amount of data that needs to be shuffled in the Reduce task begins to exceed the space available within the heap memory of each node. A similar transition from in-memory shuffling to in-disk shuffling occurs in

	# of Nodes STB / Blade	
Size	8 / 1	64 / 8
1G	49.2	49.4
2G	52.9	41.5
4G	63.9	39.6
8G	(64.5)	(48.8)
16G	(64.5)	(42.1)
32G	(61.3)	(38.3)

Table 7.2: *WordCount* Execution-time ratio.

the Embedded Cluster for smaller data sets due to the smaller memory available in the STB nodes: specifically, it occurs somewhere between 64MB and 512MB, depending on the particular number of nodes of each Embedded Cluster configuration.

Table 7.2 reports the ratios between the execution times of two Embedded Cluster configurations over two corresponding equivalent Linux Cluster configurations, for large input data sets. [2] The first column reports the ratio of the configuration with eight STBs over one single blade with eight processor cores; the second column reports the ratio of the Embedded Cluster configuration (with 64 STBs) over the Linux Cluster configuration (with eight blades for a total of 64 cores.) Across the different data sizes, the performance gap of the Embedded Cluster relative to the corresponding Linux Cluster with the same number of Hadoop nodes remain approximately constant: it is about 60 times slower for the configuration with 8 nodes and about 40 times slower for the one with 64 nodes. Notice that *these values are the actual measured execution times; they are not modified to account for the important differences among the two systems such as the 5X gap in the processor's clock frequency between the Linux blades and the STBs.* A comprehensive discussion of the reasons behind the performance gap between the two systems and how this may be reduced in the future is given in Section 7.4.5.

[2]The values in parenthesis are computed by extrapolating the execution times on the Embedded Cluster.

7.4.2 HDFS & MapReduce Benchmarks

The second group of experiments involve the execution of a suite of standard Hadoop benchmarks. The goal is to compare how the performance of the Embedded Cluster and Linux Cluster scales for different MapReduce applications. The execution times of these applications expressed in seconds and measured for different configurations of the two clusters are reported in Table 7.3. The numbers next to the application names in the first column denote input parameters, which are specific to each application: e.g., "RandomTextWriter 8" denotes that the RandomTextWriter application is running eight mappers, while the "Pi-Estimator 1k" means that Pi-estimator runs with a 1k sample size.

Sleep is a program that simply keeps the processor in an idle state for one second, whenever a Map or a Reduce task should be executed. Hence, this allows us to estimate the performance overhead of running the Hadoop framework. For the representative case of running Sleep with 128 mappers and 16 reducers, the Embedded Cluster and the Linux Cluster performance is basically the same.

RandomTextWriter is an application that writes random text data to HDFS and, for instance, it can be configured to generate a total of 8GB of data uniformly distributed across all the Hadoop nodes. When it is running, eight mappers are launched on each Linux blade, i.e., one per processor core, while only one mapper is launched on each STB node. Since the I/O write operations dominate the execution time of this application, scaling up the number of processor cores while maintaining the size of the random text data constant does not really improve the overall execution time.

Pi-Estimator is a MapReduce program that estimates the value of the π constant using the Monte-Carlo method [36]. For the Linux Cluster, the growth of the input size does not really impact the execution time for a given system configuration, while moving from a configuration with one blade to one with eight blades yields a 4x speedup. For the Embedded Cluster, in most cases scaling up the number of nodes causes higher execution times because this program requires that during the initialization phase the STBs receive a set of large class files which are not originally present in the Embedded Java Stack. This file transfer, which uses the pipelined mechanism explained in Section 7.4.4, takes a long time that more than cancel out any benefits of increasing the number of Hadoop nodes.

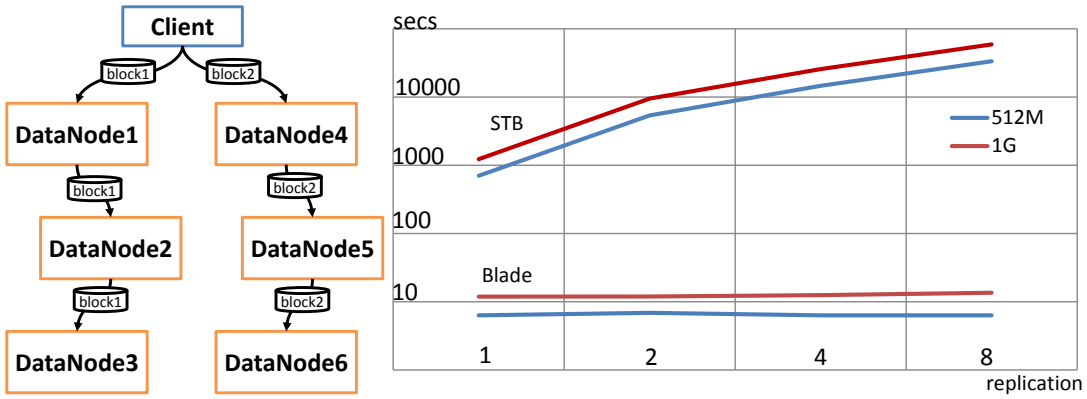
Benchmarks	8 STBs (8 cores)	64 STBs (64 cores)	1 Blades (8 cores)	8 Blades (64 cores)
Sleep	1285.1	119.6	1223.6	114.5
RandomTextWriter 8	799.6	743.9	177.6	172.0
PiEstimator 1k	461.1	163.5	212.1	52.5
PiEstimator 16k	463.4	474.0	213.7	52.5
PiEstimator 256k	603.6	783.2	214.6	52.4
PiEstimator 4M	1240.9	2048.2	213.9	52.5
PiEstimator 64M	7373.0	10482.5	314.8	58.4
K-Means 1G	3679.2	1149.3	794.7	245.0
Classification 1G	3009.0	784.9	864.7	254.5

Table 7.3: Execution times (in seconds) for various Hadoop benchmarks.

7.4.3 Data Mining Applications

To evaluate the feasibility of utilizing the Embedded Cluster system for data mining applications, we performed two experiments based on MapReduce versions of two common algorithms. *K-Means* is a popular data mining algorithm to cluster input data into K clusters: it iterates until the change in the centroids is below a threshold to successively improve the clustering result [131]. *Classification* is a MapReduce version of a classic machine learning algorithm: it classifies the input data into one of K pre-determined clusters [229]. Unlike K-Means, Classification does not run iteratively, and, therefore, does not produce intermediate data.

The last two rows in Table 7.3 report the results of running these two applications, each with an input data set of size 1GB. For both applications the results are similar: the execution time when running on the Embedded Cluster with eight STBs is about four times longer than running in the Linux Cluster with one 8-core blade; furthermore, when both systems are scaled up by a factor of eight, the performance gap grows from four to forty times. The growing gap is mainly due to the fact that scaling up the system parallelism while keeping the input data size constant leads to shuffling a large number of small data sets across the Hadoop nodes. This requires peer-to-peer communication among the nodes, an operation that the DOCSIS network of the Embedded Cluster does not support as well as the gigabit Ethernet network of the Linux Cluster does. To better evaluate the difference in transfer time between the two networks we completed the following experiment



(a) Replication pipeline. (b) Transfer time as function of replication number.

Figure 7.7: HDFS data-replication mechanism (R=3) and replication time.

focused on the HDFS data replication, which requires similar peer-to-peer communication among the Hadoop nodes.

7.4.4 Data Replication in HDFS

The Hadoop Distributed File System (HDFS) replicates data blocks through pipelining of DataNodes based on the scheme illustrated in Figure 7.7(a): for a given replication number R, a pipeline of R DataNodes is created whenever a new block is copied to a DataNode and the data are transferred to the next DataNode in the pipeline until the last one receives it. This mechanism causes a large transfer-time penalty for the Embedded Cluster due to DOCSIS-network overhead associated with the transfer of data between pairs of Hadoop nodes. Specifically, a DOCSIS network does not support direct point-to-point communications among STBs. Instead, all communications occur between a given STB and the DOCSIS router located in the cable-system head-end: this acts as a forwarding agent on behalf of the two communicating STBs. Due to this architecture, as we increase the number of STBs in the system (each STB corresponding to one Hadoop node) more slow communications between pairs of STBs occur, thus impacting negatively the overall data-replication time. In contrast, the data replication time spent in the Linux Cluster remains constant as we grow the number of nodes thanks to: (i) the fast communication channels among cores on the same blade and (ii) the gigabit Ethernet network connecting cores across different blades.

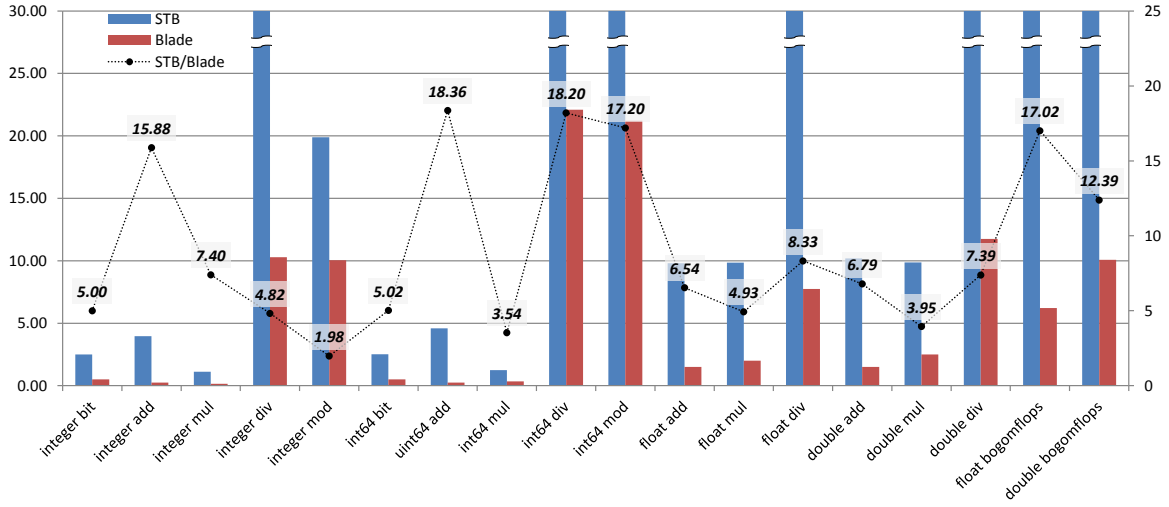


Figure 7.8: Experimental results with *lmbench* benchmark suite: comparison of the execution times of running each instruction type on the STB vs. Blade processors.

7.4.5 Discussion

The performance of executing Hadoop MapReduce applications is influenced by various system properties including: the processor speed, memory, I/O, and networking capabilities of each node. Further, the relative impact of each factor depends on the computation and communication properties of the specific MapReduce application in a way that may vary considerably with the given input problem size and the total number of nodes comprising the Hadoop system. Next, we discuss how the system properties of the Embedded Cluster compare to those of the Linux Cluster and outline how the technology trends may reduce the gap between the two systems.

Processor performance. In our experimental setup, there is a 5X gap in processor clock frequency between the Embedded Cluster and Linux Cluster nodes. Further, we empirically noticed another factor of 2X in processing speed which we attributed to the different computer architectures of the 2GHz Xeon and 400MHz MIPS processors. To quantitatively analyze the performance differences between the processors on the two clusters, we completed a set of experiments with the *lmbench* benchmark suite [219; 293], on both a STB and a blade. Fig. 7.8 shows the execution times (measured in nanoseconds) to run the different instruction types of the *lmbench* suite on one processor core of a STB and of a Blade, respectively. The dotted line over the bars indicates the value of the ratio $\frac{ExecTime(STB)}{ExecTime(Blade)}$. The Blade processor core is always faster but the relative performance

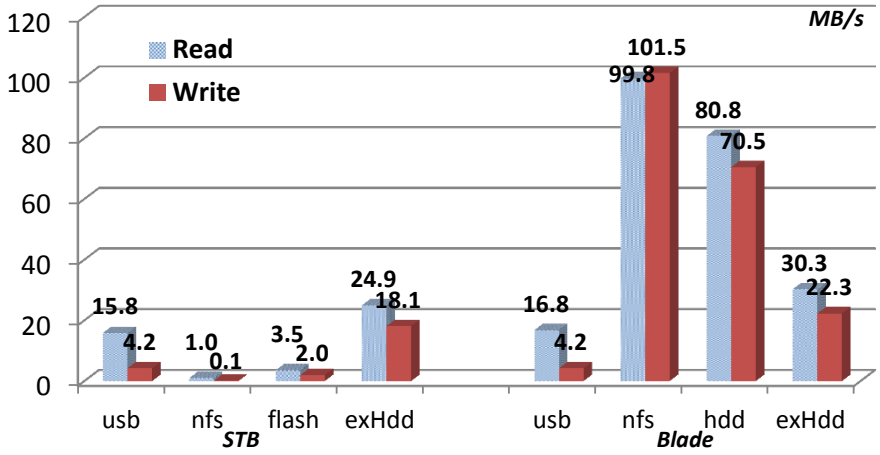


Figure 7.9: Native IO Performance Comparison

may vary: e.g., a Blade processor core is 1.98X faster than an STB processor core when they run the *integer mod* instructions, while it is 18.36X faster when they run a 64-bit *unsigned integer addition*. Besides the *integer mod instruction*, the other instructions can be categorized into two main groups: one leading to a 5-8X performance difference and another leading to a 15-18X. The performance gap is partially due to the more sophisticated architecture of the Blade processor and partially to the clock frequency which is 5X slower for the STB processor. This part of the gap is expected to decrease considerably in the near future as next-generation STB devices will incorporate commodity 1GHz+ multi-core processors, similar to those found already in consumer electronic devices such as smartphone and tablets, while it is unlikely that the clock frequency of the Blade processor will increase considerably.

I/O Operations. The *RandomTextWriter* benchmark represents many MapReduce applications which execute numerous data-block storage operations. In fact, the Hadoop system itself can be very I/O intensive when performing data replication. We run the *TestDFSIO* test to evaluate the I/O performance of HDFS by reading/writing files in parallel through a MapReduce job. This program reads/writes each file in a separate map task, and the output of the map is used for collecting statistics relating to the file just processed; then, the statistics are aggregated in the reduce task to produce a summary. The results of running TestDFSIO reveal that an STB has 0.115MB/s reading and 1.061MB/s writing speed while the corresponding values for a Linux blade are 68.526MB/s and

99.581MB/s. [3] We also run a simple native C program that executes read/write operations using large files on the two clusters with four different interfaces: USB, FLASH, NFS, and HDD. The results are reported in Figure 7.9. We note that the network performance of STB NFS reads/writes is significantly less, by a factor of nearly 100, than the network performance of the Linux blade server. This gap is primarily due to the DOCSIS network, whose effective transfer rate is limited to 4MB/s compared to 1Gb/s Ethernet network, whose effective maximum transfer rate is closer to 125MB/s. On the other hand, the measured performance of the USB and external hard-drive interfaces on both the STB and Linux blade server is comparable. This is due to the common commodity SoC for USB and disk interfaces used in the design of both the STBs and blades. In our experiments, the Linux blades use an internal hard-drive disk (HDD) while the STBs, which do not contain an internal hard-drive, rely on a USB memory stick whose read performance is six times slower (and write performance is 24 times slower) than the HDD when providing HDFS storage. This gap can be reduced by having the STBs use a better file system for the USB sticks than FAT32 such as SFS [222]. Also, as shown in Figure 7.9, an external USB HDD could provide a 1.5-4.2 speed-up for reading/writing over the USB memory stick. Here, the technology trends should provide next-generation STB devices with HDD and USB 3.0.

Networking. The lack of support for peer-to-peer communication among STBs in the DOCSIS network limits considerably the HDFS replication mechanism (as discussed in Section 7.4.4), the Hadoop shuffling operations (as seen for the K-Means, Classification and WordCount programs), and the transfer of large class files during the initialization phase (as in the PI-Estimator). In particular, shuffling generates an implicit all-to-all communication pattern among nodes that is application specific: each node sends its corresponding Map results to other nodes through HTTP, generating $|Node|^2$ communication exchanges, which for the DOCSIS network results in inefficient upstream communication requests as the nodes attempt to transfer data blocks from Mappers to Reducers. A similar performance impact occurs during Hadoop replication: for a given replication factor R and a total number of blocks M, the number of DOCSIS upstream communication transfers to complete replication is $M \times (R-1)$. As the input size increases the number of blocks increases in direct proportion, thus increasing the replication time. The scalability in Embedded Cluster largely depends

[3] The STB shows significant difference between upload and download speed due to the inherently asymmetric and lower transfer rate characteristics of the DOCSIS network.

on the amount of data to be shuffled generated by the Map tasks and the replication communication overhead. This problem may be addressed in part with the deployment of the higher performance DOCSIS 3.0 standard [114], which supports up to 300 Mb/s upstream bandwidth. Then, opportunities for further improvements include: optimization of the Hadoop scheduling policy, network topology optimization, and leveraging the inherent multi-casting capabilities of DOCSIS to reorder the movement of data blocks among nodes and reduce network contention.

7.5 Related Work

The Hadoop platform for executing MapReduce applications has received great interest in recent years as problems in large-scale data analysis and Big Data have increased in importance. Work in the area of heterogeneous MapReduce computation, however, remains rather limited, notwithstanding the growth of embedded devices interconnected through broadband networking to distributed data centers. Our work is aligned with efforts in the Mobile Space to bridge MapReduce execution to embedded systems and devices. For example, the Misco system implements a novel framework for integrating smartphone devices for MapReduce computation [88]. Similarly, Elespuro *et al.* developed a system for executing MapReduce using smartphones under the coordination of a Web-based service framework [90]. Besides the fact that our system uses a wired network of embedded stationary devices instead of a mobile network, the main difference with these systems is that we ported the Hadoop framework, including the HDFS, based on the Java programming model. Other related work includes utilizing GPU processors to execute MapReduce [134]. While most related work in adapting MapReduce execution to embedded devices has focused on leveraging service-side infrastructure, our work is closer to current research under way for large scale execution of MapReduce applications on the Hadoop platform across Linux blades and PC clusters [304].

7.6 Concluding Remarks

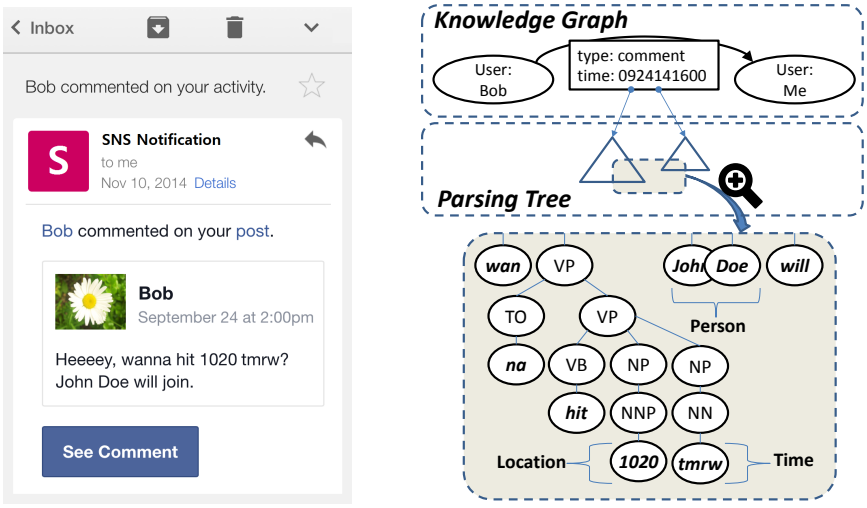
We developed, implemented, and tested a heterogeneous system to execute MapReduce applications by leveraging a broadband network of embedded STB devices. In doing so, we addressed various general challenges to successfully port the Hadoop framework to the embedded JVM environment. We completed a comprehensive set of experiments to evaluate our work by comparing various con-

figurations of the prototype Embedded Cluster with a more traditional Linux Cluster. First, the results validate the feasibility of our idea as the Embedded Cluster successfully executes a variety of Hadoop applications. From a performance viewpoint, the Embedded Cluster typically trails the Linux Cluster, which can leverage more powerful resources in terms of processor, memory, I/O, and networking. On the other hand, for many applications both clusters demonstrate good performance scalability as we grow the number of Hadoop nodes. But a number of problems remain to be solved to raise the performance of executing MapReduce applications in the Embedded Cluster: in particular, critical areas of improvement include the STB I/O performance and the communication overhead among pairs of STBs in the DOCSIS broadband network. Still, the gap between embedded processors and blade processors in terms of speed, memory, and storage continues to decrease, while higher performance broadband networks are expected to integrate embedded devices into the Cloud. These technology trends hold the promise that future versions of our MapReduce computing system can help to leverage embedded devices for Internet-scale data mining and analysis.

Chapter 8

Algorithm-Division Reverse Offloading: Locally Customized Training

Personal mobile devices offer a growing variety of personalized services that enrich considerably the user experience. This is made possible by increased access to personal information, which to a large extent is extracted from user email messages and archives. There are, however, two main issues. First, currently these services can be offered only by large web-service companies that can also deploy email services. Second, keeping a large amount of structured personal information on the cloud raises privacy concerns. To address these problems, together with Karl Stratos and Luca Carloni, I developed LN-Annote, a new method to extract personal information from the email that is locally available on mobile devices (without remote access to the cloud). LN-Annote enables third-party service providers to build a question-answering system on top of the local personal information without having to own the user data. In addition, LN-Annote mitigates the privacy concerns by keeping the structured personal information directly on the personal device. Our method is based on a named-entity recognizer trained in two separate steps: first using a common dataset on the cloud and then using a personal dataset in the mobile device at hand. We optimized LN-Annote by implementing an OpenCL version of the custom-training algorithm to leverage the Graphic Processing Unit (GPU) available on the mobile device. We also show how we developed the MCC applications in the LN-Annote system, including the Android app for local training, and assessed the optimization technique by testing its scalability and performance based on the tool NETSHIP.



(a) An SNS Notification Email (b) Knowledge Graph and Parse Tree

Figure 8.1: Parsing email to collect personal information.

8.1 Introduction

Recent advancements in personalized web-based services have enriched our daily lives. Intelligent personal assistant services such as Google Now [13] or Apple's Siri [5] can give "directions to home" or alert that "it's time to leave for your next meeting". Meanwhile, personal search services can answer queries based on the user's personal information. Googling "my flights", for instance, produces the upcoming flight reservations that the user has made. Personalized advertisement is another important (and most profitable) instance of personalized web services. The advertisement systems of Amazon, Facebook and Google [2; 10; 11] are known to utilize viewers' personal information such as previous purchase history.

What makes all these personalized services possible? The personal information collected by the service providers. Since its quality determines the quality of the personalized services, web service providers put in significant efforts to improve and extend its collection. One vast source of personal information is found in users' emails. Large web-service companies that provide also email services (like Google, Microsoft, and Yahoo) have the means to offer rich personalized services precisely thanks to the personal information they extract from the users' emails. The example of Figure 8.1 illustrates this process. A notification email of a message posted on a Social Network Service (SNS) account by one of the user's friend is parsed through a sequence of steps to build structured data,

including: 1) a knowledge graph indicating the subject, the object, a type of the action, and the contents of the comment; 2) the parsing tree of the comment; 3) various grammatical tags such as part-of-speech (e.g., Verbal Phrase and Noun Phrase) and Named-Entity Recognition (NER) labels (e.g., Person, Location, and Time). This kind of structured personal information is stored and used later in various ways: e.g., to improve personal search services by retrieving results that are relevant to the named-entities related to the user.

Thanks to the growing amount of personal data that are available to be collected, it is easy to predict that personalized services will continue to evolve and expand. There are, however, limitations and concerns. First, the current methods of information extraction are not feasible for any small company that doesn't have its own email service because they are based on accessing large data sets collected with proprietary email services. Second, keeping large amount of structured personal information on centralized remote cloud servers raises privacy and security concerns [64; 275].

To address these problems, we present LN-Annote (Locally customized NER-based Annotation), a novel information-extraction subsystem that is designed and optimized to process the email data available *locally* on each personal mobile device. Our contributions include:

- a distributed learning model based on two phases: *universal training* to generate a common parameter set on the cloud and *custom training* to refine and optimize the shared common parameter set by using the email data locally available on each mobile device;
- a discussion on how to extend the architecture of a personal search system to integrate the LN-Annote subsystem;
- an implementation of LN-Annote using locally available information and optimization methods leveraging the GPU on the mobile device; and
- an extensive set of experimental results to prove the feasibility, effectiveness, and efficiency of our approach.

In Section 8.2 we describe a personal search service as an example of a personal information system where LN-Annote is employed as a subsystem. In particular, we compare two different approaches for information extraction, on the cloud and on the mobile. Also, we illustrate the workflow of LN-Annote to extract personal information from emails stored on smart devices, which can

then be available for many personalized service providers. In section 8.3 we show how LN-Annote is designed on the NETSHIP framework. In Sections 8.4 and 8.5 we present how we implemented and optimized our system. In Section 8.6 we present a comprehensive set of experiment results to show the efficiency and the effectiveness of our approach. In particular, we show the advantages provided by the addition of custom training on top of universal training.

8.2 LN-Annote System Design

In this section we present the design of LN-Annote, its main components, and how these interact with other modules as part of a bigger system. LN-Annote is an NER-based system optimized to extract information from email messages, in particular those sent via SNSs. The focus on email is motivated by two observations: 1) email messages convey scads of personal information and 2) many web services send notification emails with very useful data such as reservations or recommendations.

8.2.1 Information Extraction and NER

Nowadays many companies send emails to their customers for various purposes such as discount offers, purchase history, appointment reminders, or activity updates on SNSs. As more companies integrate their services with email systems, these email messages contain a growing amount of personal information. Meanwhile, more and more people use their email to manage personal information [341]. Hence, the ability to extract personal information from emails becomes increasingly important. Nonetheless, existing extraction techniques have various limitations. One approach is to write vendor-specific parsing scripts; this, however, requires a large amount of manual labor to update the scripts whenever the vendor changes the email format. Another approach is to use Microdata embedded in the emails containing structured information [135]; this, however, is currently not very effective because the number of email messages that contain Microdata is very limited.

To overcome these problems, Natural Language Processing (NLP) techniques have been proposed to assist service providers in extracting useful information [243; 316]. Named-Entity Recognition (NER) is a popular NLP technique to classify given vocabularies into predefined categories [38; 240]. A wide variety of systems use NER for different text types such as queries, SNS posts, or

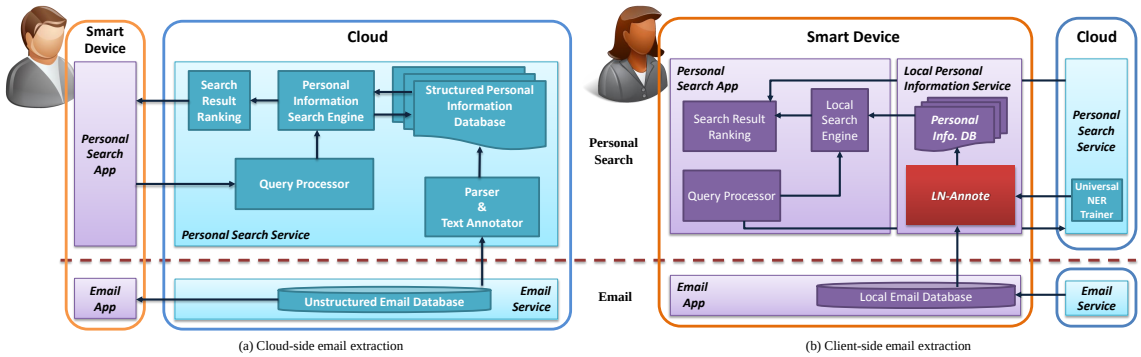


Figure 8.2: An architectural comparison between two personal search systems.

résumés [129; 201; 247].

The performance of a NER system can be evaluated using various metrics. One of the most widely used metric is the F_1 *score* which is defined as the harmonic mean of *Precision* and *Recall*:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8.1}$$

Precision, or *Positive Predictive Value*, is the correctness of the predicted classification and Recall, or *True Positive Rate*, is the coverage of the positive cases.

8.2.2 A Use Case: Personal Search Service

The service infrastructure that we present here is an example of a system where LN-Annote works as a key component subsystem in collaboration with other components.

A personalized search service provides answers based on the personal information that it has collected from the emails of the user who requested the query. The personal information of each user is collected periodically from the email database and stored into a structured database to simplify its retrieval. The diagrams of Figure 8.2 illustrate two different approaches to implement this service. Each shaped object represents a processing component or a document database and each arrow indicates a direction of data flow. The users access the services through their smart devices, searching personal information through a *Personal Search App* and receiving emails through an *Email App*.

Figure 8.2(a) shows an approach where the extraction happens in the cloud. Periodically, e.g., once a day, the extraction system accesses directly the email database to update the *Structured Per-*

sonal Information Database. When a user issues a personal query with the *Personal Search App*, the query is passed to the personal search where it is handled by the *Query Processor* to disambiguate it and augment it. The processed query is then passed to the *Personal Information Search Engine* which retrieves relevant documents from the Structured Personal Information Database. The retrieved documents are ranked by the *Search Result Ranking* system and returned to the particular app running on the mobile device, e.g., the Personal Search App, so that the user can see the results. This approach, however, has several disadvantages. First, it is feasible only for a very small group of service providers that own email services with a sufficient number of users. Hence, many service providers that do not have access to email services miss a major source of personal information. Second, it raises privacy and security concerns over the extraction, storage, and processing of very large amount of personal information in centralized remote cloud servers [185].

To address these challenges we propose the approach illustrated in Figure 8.2(b). In this approach, the personal information extraction becomes a task that runs locally on the personal device of each user, where recently fetched emails are stored by the email app.[1] Our proposed LN-Annote in *Local Personal Information Service* is similar to the *Parser & Test Annotator* of Figure 8.2(a) but uses NER and creates *Personal Information Database* on the local device. As shown in Fig 8.2(b) the local extraction of personal information resolves the dependency between the personalized service and the email service, thus allowing personalized service providers without their own emails services to access the local personal information database. Also, since the information remains local in the mobile device, privacy concerns and security issues are effectively alleviated [312]. Finally, the introduction of LN-Annote, a distributed NER subsystem, improves considerably the extraction accuracy while reducing the computation burden on the cloud servers (as discussed in more detail in Section 8.7). On the other hand, our approach requires that the mobile devices perform some additional amount of computation and this may have a negative impact on their overall performance and energy consumption. To minimize this impact, we have developed a method for custom training based on feature templates (as described in Section 8.5.1) and we have parallelized the key

[1] Accessing other app's database is not trivial on smart platforms where each app runs in its own sandbox environment, but it is still possible through a couple of options. We have implemented email apps that share email data with other allowed apps through inter-app communication methods allowed on each platform, e.g., *ContentProvider* on Android [12]. The users can control which apps are allowed to access the emails.

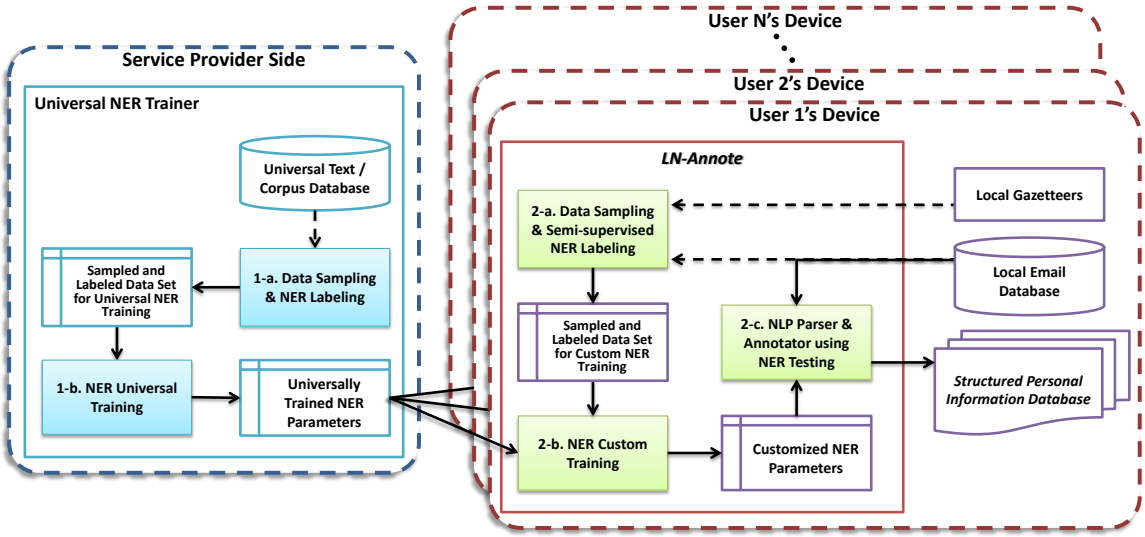


Figure 8.3: The flowchart of LN-Annote.

algorithms to run on the GPU present in each modern mobile device (Section 8.5.2).

8.2.3 The LN-Annote System Workflow

In this section, we describe the LN-Annote subsystem that we built to implement the approach shown in Figure 8.2(b). To achieve more accurate prediction, we conceived a novel method that performs training for NER in two separate main steps, as illustrated in Figure 8.3 (in this flowchart, a solid arrow represents a flow of data and a dashed arrow represents a sampling activity.) The first step, shown in the blue box (left), is *universal training*: this is essentially identical to traditional learning and returns learning parameters that will be **shared among all users**. The second step, shown in the red boxes (right), is *custom training*: it runs on each personal device, takes the learning parameters from the universal training, and enhances them by further training with the **locally accessible dataset which is specific to each user**. The main goal of our method is to produce locally customized learning parameters that work well for the particular local environment. However, the parameters need to perform well also on the global texts, by preserving the knowledge from universal training.

As shown later, LN-Annote achieves extraction performance comparable to training for a combination of the global dataset and the personal dataset, while requiring a significantly smaller amount of computation. Next, we provide more details on the two main steps.

1. Universal Training in the Cloud. This step consists of two substeps:

1-a. *Data sampling and NER labeling* is a preparation activity that takes samples from a universal text database and creates labels for the sampled data to feed the supervised training of Substep 1-b. Choosing a representative dataset with an appropriate amount is an important task for the quality of the training [236]. The sampled data can be labeled using different methods: a) manual labeling, b) manual labeling and running semi-supervised learning, and c) running unsupervised learning [93; 317]. For the experiments of this research, we used the CoNLL03 dataset provided with manually labeled NER tags [309], while semi-supervised learning is commonly used for large datasets.

1-b. *NER universal training* uses supervised learning to process the labeled data produced by Substep 1-a. In traditional machine learning, the learning parameters created by this kind of algorithm are directly used to test the prediction of NER labels for the actual dataset. In our approach, instead, these learning parameters are shared with the multiple mobile devices for custom training.

2. Custom Training & Testing on the Mobile Device. This step consists of three substeps:

2-a. *Data sampling & semi-supervised NER labeling* works similarly to Substep 1-a to produce labeled data for Substep 2-b. Here, the inputs are text samples selected from the local email database used by the email app running on the mobile device. Also, in this case the manual labeling cannot be applied because it is infeasible to ask the user of the personal device to do it. Instead, we obtained *local gazetteers*, a list of named-entities from a reliable source [231]. This substep is performed automatically by using a semi-supervised learning algorithm based on the labels from the gazetteers. How to obtain gazetteers is explained in Section 8.4.3.

2-b. *NER Custom Training* updates the NER parameters by learning from the labeled dataset generated by the emails on the mobile device. The use of updated parameters is expected to keep the same performance as the use of universally trained parameters on the global dataset, while delivering better performance on the local emails.

2-c. *NLP Parser & Annotator using NER*, the final substep of LN-Annote processes the emails on the mobile device. This is done using NER based on the parameters created in the Substep 2-b. The outcome is stored in the Structured Personal Information Database where it can be used by any personal services running on the mobile device, as long as this is allowed by the user.

Notice that each substep of LN-Annote occurs with a different frequency: universal training (Substeps 1-a and 1-b) is done only once on the cloud servers; custom training (Substeps 2-a and

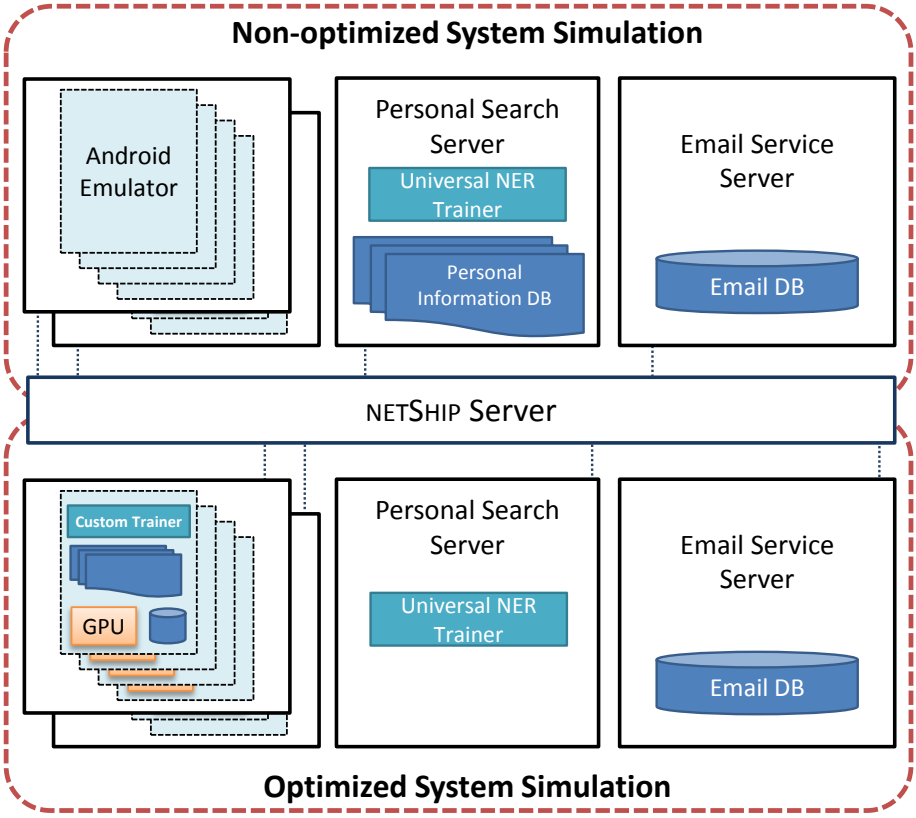


Figure 8.4: Simulating the LN-Annote system using NETSHIP.

2-b), i.e., obtaining gazetteers, takes place periodically, e.g., once a week or a month; and finally, the actual information extraction (Substep 2-c) runs rather frequently, e.g., everyday.

8.3 Designing LN-Annote on NETSHIP

We develop LN-Annote using the tool NETSHIP introduced in Chapter 4. Figure 8.4 presents the configurations of simulating the LN-Annote system using NETSHIP. The LN-Annote system can be optimized by leveraging the mobile GPU present in the mobile devices as studied in Chapter 8. Thus, there are two sets of simulation instance configurations: one for the simulation of the system before the optimization technique is applied (as shown in the top half of the figure) and another for the simulation of the system when the proposed optimization technique is adopted (as shown in the bottom half of the figure).

In each simulation configuration set, there are three types of virtual machine instances that are

used to implement the system on top of the NETSHIP server instance, which manages the simulation including the synchronization. The first type of virtual machine instances execute multiple Android emulators (the left-most part of the figure). Each of these virtual machine instances can incorporate up to four Android emulator instances. In the non-optimized system simulation case, these Android emulators run an Android app that runs only the personal search client service (the top-left boxes). In the optimized system simulation, the app also executes local training and information extraction functions utilizing the virtual GPU models that multiplex the host GPU (the bottom-left boxes). Second, the personal search server instance is where the universal NER trainer runs (the middle part of the figure). The personal information database resides in the non-optimized case (the top-middle box). As a result of local training and information extraction, it keeps only the universal NER trainer when optimized (the bottom-middle box). This instance does not necessarily have to be scalable for the experiments in this Chapter. However, the large-scalability feature offered by NETSHIP allows also this instance to easily scale up in case the size of the database outgrows the provided virtual machine instance, which might be needed for the higher scalability as the number of users grows in the future. Third, the email service server instance behaves as an external email service provider where the email database is stored (the right-most part of the figure). This virtual machine provides the email content to the universal NER annotator on the personal search server when the system is not optimized (the top-right box) and to the client app when the system is optimized (the bottom-right box).

We use NETSHIP to develop the client-side information extraction app for Android and to test the scalability of the system. Throughout the simulation of the LN-Annote system based-on NETSHIP, We are able to perform the following experiments:

- Developing the Android app before having a physical device at hand.
- Testing the scalability of the server for the app beyond the number of physical devices we possess and therefore saving the budget for the tests.
- Comparing the performance of the non-optimized configurations and the optimized configurations and thus efficiently finding the best optimization technique.
- Calculating the estimated power consumption of each implementation.

8.4 NER System Implementation

We designed the LN-Annote system to work with different types of incremental learning algorithms [181]. For instance, it can work as a framework running the universal training in the cloud and the custom training on the mobile while sharing the learning parameters between the two. In this section, we introduce as a showcase our *Neural Network* model for NER.

8.4.1 NER Algorithm Selection

NER systems may adopt a linguistic grammar-based model or a statistical learning model. We developed a system using a window-based neural network model for NER [75; 102]. Neural network learning algorithms learn the representation of multiple layers with increasing complexity and abstraction. A neural network works iteratively through *feed-forwarding*, a process to obtain the hypothesis and the cost objective (error), and *back-propagation*, a process to adjust the parameters according to the error.

Compared to various alternatives, neural network models are known to be difficult to understand with respect to the internals of the parameter optimization process. For this reasons, in many state-of-the-art NER systems programmers use other machine learning algorithms such as Conditional Random Fields [102; 319], Maximum-Entropy Markov Model [105], or Perceptron [74], which in many cases achieve pretty high F_1 score (over 90). Still, we chose to build a neural network model for NER for the following reasons. First, a neural network automatically chooses feature values through feed-forwarding and back-propagation. This makes the system free from the need of using carefully hand-crafted features that are required by other learning techniques. In particular, in our system it enables the automation of the preparation and training for the customized learning step. Second, neural network algorithms make heavy use of linear algebra and, particular, matrix operations: these can be easily parallelized and executed on the GPU of the mobile device, thereby reducing the time and energy spent on the computation in the custom training. The idea of using the mobile GPU for acceleration translated into the needs for simulation tools that can support simulations of mobile GPUs and motivated the development of ΣVP, which we discuss in Chapter 5.

Neural network models are prone to *catastrophic forgetting* (or catastrophic interference), an extensive loss of previously learned information that may occur while learning new training data. This problem has its main causes in the representational overlap within the learning system and

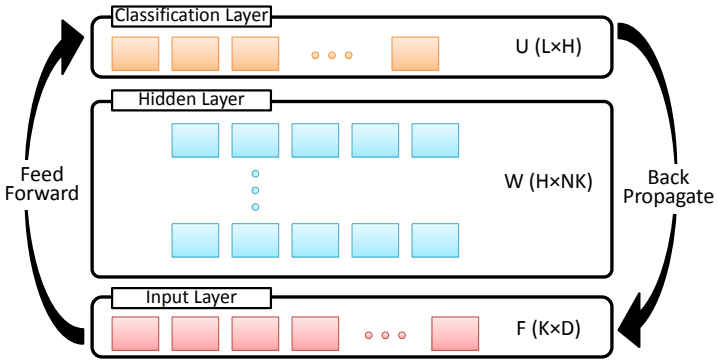


Figure 8.5: The designed neural network architecture.

in the side-effects of the training methods used (rather than in the lack of sufficient computing resources in the system) [77]. In our system, catastrophic forgetting could manifest in a particular way: the customized NER parameters can recognize entities learned during custom training while losing the ability of identifying entities previously learned during universal training. There are several known solutions to mitigate catastrophic forgetting, including: conservative training and support vector rehearsal [34], multi-objective learning [159], sweep rehearsal [267], unsupervised neuron selection [122], fixed expansion layer [76], and ensemble of classifiers [252]. In our neural network model we avoid catastrophic forgetting by designing an incremental neural network model that introduces a scale factor to the weight adjustment [108; 329]. This leads to excellent results, as discussed in Section 8.6.

8.4.2 A Neural Network Model for NER

Both universal training and custom training are based on supervised learning and use the same neural network model. In fact, the two training algorithms are essentially the same for consistency and compatibility of network parameters. The only differences are the number of iterations and the learning rate. We use a window-based network model with a fixed window size. This means that, while labeling a given word, a few nearby words are also processed to provide a local context that helps capturing the characteristics of the target word.

In the neural network we have three layers, as shown in Figure 8.5: the input layer, the hidden layer, and the classification (output) layer. The input layer consists of feature vectors, or word representations, obtained from the unsupervised pre-training [145]. Each feature vector consists of K float values to represent a vocabulary in the dictionary of D vocabularies. The hidden layer, with

a dimension of H, is used to derive hidden features by computing the input layer and the weight vectors. The dimension of weight vectors W for the hidden layer is $H \times NK$ where N is the size of window. On top of the hidden layer, there is the output layer for *softmax regression* to predict named-entities [89]. The weight vectors U for the output layer is of size $L \times H$, where L is the number of possible output classes.

Feed-Forwarding. Feed-forwarding in neural networks is a process to compute prediction values. We first calculate $z_i = Wx_i + b_1$ where x_i is the feature vectors of the words in the i-th window and b_1 is a bias vector. Then, we obtain $a_i = f(z_i)$ by applying a nonlinearity function f. Finally, we compute the hypothesis $h_i = g(U^T a_i + b_2)$, where b_2 is a bias for the softmax regression; the sigmoid function g makes the regression result fit smoothly into a classification result. The final prediction, h_i, is the probability that x_i is a named-entity of each class. Here, along with the feature vectors used as the input layer, the weight vectors and the bias vectors are considered as the NER parameters of Figure 8.3.

Algorithm Configuration. We use the hyperbolic tangent function $tanh$ as the nonlinearity function f. Note that its derivative can be expressed with the $tanh$ function itself, i.e., $f'(x) = \frac{d}{dx}\tanh x = 1 - \tanh^2 x$. This greatly simplifies the back-propagation, thereby reducing the amount of computation for training and testing on the mobile side. The sigmoid function we use is $g(z) = \frac{1}{1-e^{-z}}$.

Training Objective. In both the universal training and the local training we minimize the same cost (objective) function which is the following cross-entropy error with a regularization factor:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{L}\left[1\{y^{(i)} = j\}\log\left(h_\theta^{(i)}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$$

where the model parameter θ includes the input values extracted by the current window, λ is a weight decay term (for any $\lambda > 0$) introduced for regularization, and $1\{condition\}$ is a function that returns 1 when $condition$ is $true$ and 0 otherwise. During back-propagation, we minimize this objective function using stochastic gradient descent, a first-order optimization method widely used for training a variety of models in machine learning.

Platform	API
Android	*ContactsContract.Contacts.CONTENT_URI*
iOS	*ABAddressBookCopyArrayOfAllPeople*
Windows Phone	*Microsoft.Phone.UserData.Contact.SearchAsync*

Table 8.1: Mobile APIs for *PER* entities.

Provider (Base URI)	REST API (PER) (ORG/LOC/MISC)
Facebook	*/v2.1/me/friendlists*
http://graph.facebook.com	*/v2.1/me/feed?with=location*
Google+	*/v1/people/me/people/connected*
https://www.googleapis.com/plus	*/v1/people/me/activities/public*
Twitter	*/1.1/followers/ids.json*
https://api.twitter.com	*/1.1/statuses/user_timeline.json*

Table 8.2: SNS open APIs for various entities.

8.4.3 Local Gazetteer from Mobile

Differently from universal training, which can benefit from the manually tagged corpus, the custom training on mobile devices cannot rely on any human effort to label the dataset. To automate the custom training while obtaining a high-quality training dataset, we used the idea of gazetteers [231] and developed a method to acquire gazetteers from local and external sources. Gazetteers are a named-entity list obtained from an external, reliable source. Our gazetteer-induction method uses the contact list, checked-in locations, and liked pages through the mobile platform and SNS open APIs. The available APIs for collecting this gazetteer information from mobile and SNS platforms are listed in Table 8.1 and Table 8.2, respectively. The labeled dataset is then fed into the semi-supervised learning algorithm to create a training set for the custom training.

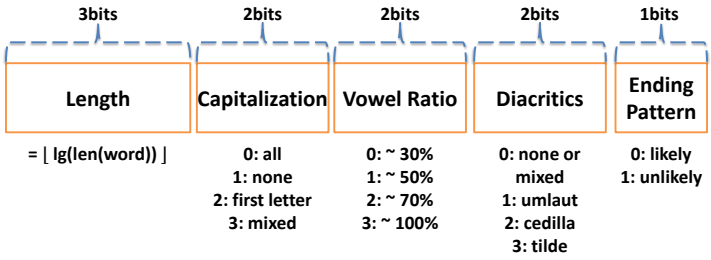


Figure 8.6: The index of grouped feature templates.

8.5 Optimization

The execution of custom training on the personal mobile device poses some challenges in terms of increased energy consumption and the possible slowdown of the overall device. To address these challenges we performed two major optimizations.

8.5.1 Feature Templates

Our neural network model for custom training is designed for enhancing accuracy by using the information available on the local device. Hence, it is important to capture the peculiar vocabularies observed during the custom training on the device. For example, non-standard words (frequently used on the web or in text messages as shown in Figure 8.1) or abbreviated words (such as 'CROACC' [2]) are less likely to be part of a general dictionary and, therefore, less likely to have occurred during universal training. They, however, may occur frequently in the custom training with the database of a particular user. If so, for this user these words may be named-entities or may be useful to determine nearby named-entities.[3] Thus, we capture the newly discovered words and add them to our vocabulary list, while also creating feature vectors for these new words. A performance concern, however, arises if we initialize the new feature vectors with random values. We keep the learning rate low in the custom training because it is additional training on top of the universal training and the scale factor of incremental back-propagation limits the learning rate to prevent catastrophic forgetting. Starting from parameters with the random values require more learning iterations, which slows down the training. To address this challenge, we developed

[2]Cannot Rule Out Anything, Correlate Clinically.

[3]Notice that this is different from the case in universal training when newly discovered training vocabularies that have no corresponding feature vectors are discarded because they appear rarely.

a new technique to assign initial values to the feature vectors for the newly discovered words in the custom training. This technique copies template feature vectors from a group to which a new word belongs. Then, further training can converge with less learning iterations while keeping the learning rate low. As shown in Figure 8.6, we use local features for grouping each word [202; 310]. The *Ending Pattern* is a boolean value representing if the word ends with one of the predefined postfixes, such as "ie", "son", "ng", or "a". These local features are then encoded to calculate the template group index for feature vectors, with a maximum index of $2^{10} - 1 = 1023$. For instance, the group index of a new word (*YoungHoon*) is 488. Then, the feature vectors for the word are copied from the template at index 488.

8.5.2 Hardware Acceleration

While feature templates can reduce the number of iterations to learn new vocabularies, the largest performance bottleneck still lies in the training algorithm. This leads into problems including: high energy consumption, CPU occupation, and delay in using the training results. In previous works, NER algorithms were accelerated by using some hardware units available on computer servers [95; 239]. Our goal, however, is to remove the performance bottleneck during the custom training on the mobile devices.[4] Furthermore, mobile devices are particularly prone to these problems due to their relatively slower CPUs and more limited energy budget. To accelerate the NER custom training and testing algorithm we exploit the GPU on the mobile device. Besides performance gains, GPU-based acceleration can also lower energy consumption due to the decreased execution time compared to the execution on the device CPU.

The implemented OpenCL kernel executes the entire training algorithm, e.g., window index extraction, hypothesis computation, and gradients calculation. Listing 8.1 shows an OpenCL kernel that computes the hypothesis of the softmax regression (the top layer in Figure 8.5) for a NER class.[5]

[4] In most cases, training requires more computation than testing for the same size of dataset.

[5] This implementation is from our second version where each kernel computes a small portion of the algorithm.

```
/* Put product of global values to local mem */
if (gid * 4 < hiddenSize) {
  partial_dot[lid] = U[gid] * a[gid];
}
else {
  partial_dot[lid] = 0;
}
barrier(CLK_LOCAL_MEM_FENCE);

/* Repeatedly add values in local memory */
int nElem = group_size;
int i = (nElem + 1) / 2;

do {
  if(lid < i && lid + i < nElem) {
    partial_dot[lid] += partial_dot[lid + i];
  }
  barrier(CLK_LOCAL_MEM_FENCE);

  nElem = i;
  i = (i + 1) / 2;
} while (i != nElem);

/* Transfer final result to global memory */
if(lid == 0) {
  h[get_group_id(0)]
    = dot(partial_dot[0], (float4)(1.0f));
}

barrier(CLK_GLOBAL_MEM_FENCE);

if (gid == 0) {
  for (int i = 1; i < get_num_groups(0); i++) {
    h[0] += h[i];
  }

  h[0] += b2;
  h[0] = 1.0 + pow((float) M_E_F, -h[0]);
  h[0] = 1.0 / h[0];
}
```

Listing 8.1: A portion of OpenCL kernel that computes the hypothesis.

This kernel implementation uses a technique that first calculates the dot products of sub-matrices on the local memory storage shared among GPU thread blocks and then it sums up these partial dot products to quickly compute the dot product of the entire matrix. This exploitation of the local memory speeds up the summation process because for a GPU core accessing the local memory is faster than the global memory [349].

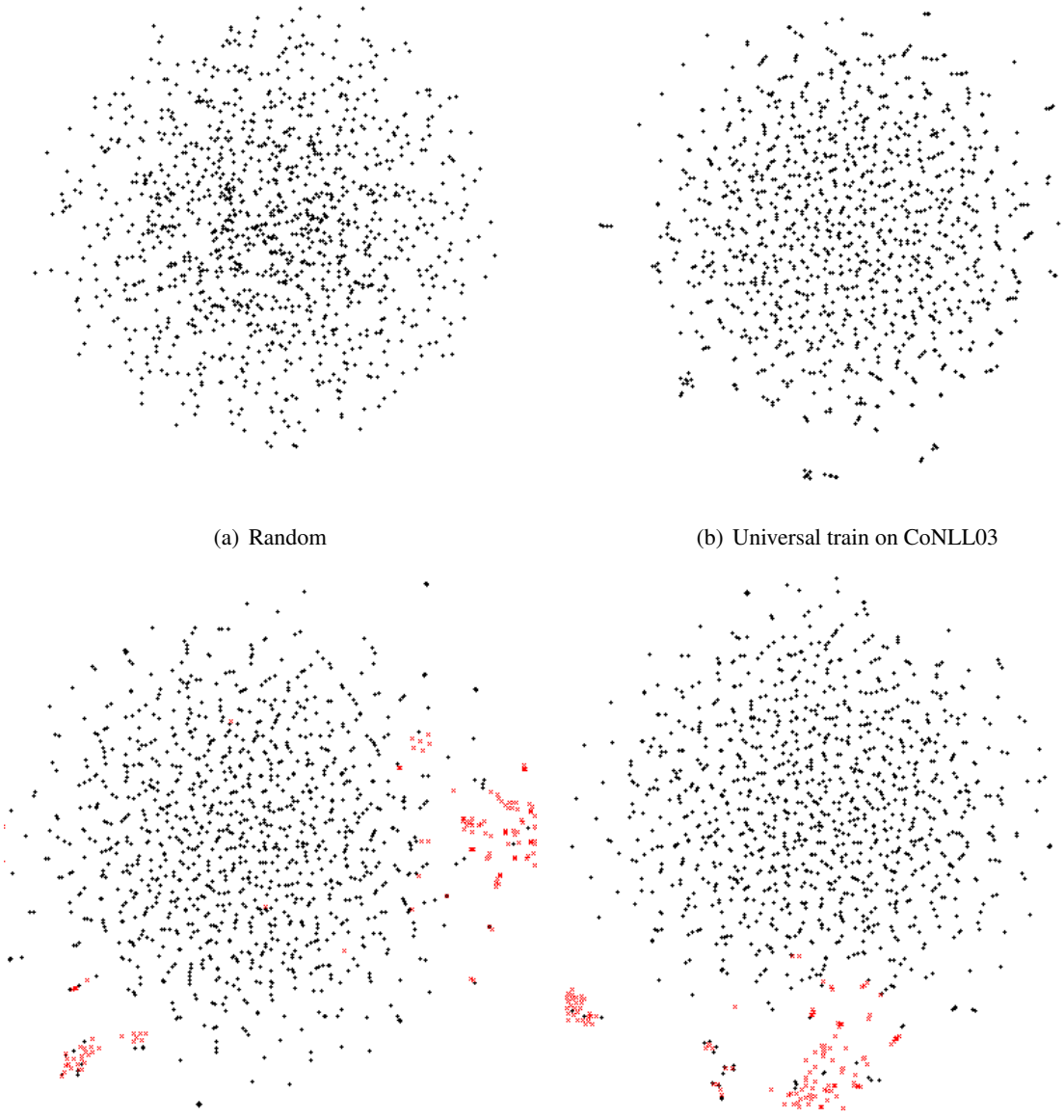
8.6 Experiments

In the following experiments, we used the CoNLL03 shared task [309] in the universal training while we leveraged emails dataset in the custom training. CoNLL03 provides an English training dataset (*CoNLL03_train*) and two English test datasets (*CoNLL03_testa* and *CoNLL03_testb*). The actual email datasets used in the experiments were created with SNS notification emails, such as the example presented in Figure 8.1(a), chosen from personally donated emails for research purposes. We created a training dataset and a testing dataset for each user and used these in Substeps 2-b and 2-c of Figure 8.3, respectively. Let *email_traina* denote the training dataset from user$_a$ and *email_testa* denote the testing dataset from the same user. While a semi-supervised learning and gazetteers were used with the email datasets for training as explained in Section 8.2.3, the email datasets for testing were labeled following the CoNLL03 guidelines [309] for the experiments in this section. Each email dataset for custom training consists of 128,000 words out of which approximately 8,000 to 21,000 words are unique within that dataset. The used email dataset for testing contains around 64,000 words. Each measured value that has a possibility of variation, such as execution time and power consumption, was executed ten times and averaged excluding the largest observation and the smallest observation.

8.6.1 Learning NER Feature Vectors

In this section, we compare how the feature vectors, the main learning parameter in our NER neural network model, change as we execute universal training and custom training.

One of the difficulties in using a neural network model is understanding its internal behavior. In many cases, visualizing how the learning iterations progress can help developers understand and improve the system. t-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear method



(a) Random

(b) Universal train on CoNLL03

(c) Custom train on email_traina

(d) Custom train on email_trainb

Figure 8.7: t-SNE plots of feature vectors.

	Training			Testing		
	Prec.	Recall	$F_{\beta=1}$	Prec.	Recall	$F_{\beta=1}$
LOC	84.28%	80.02%	82.10	83.15%	75.87%	79.34
MISC	79.24%	72.45%	75.69	73.86%	72.34%	73.10
ORG	85.58%	84.56%	85.07	78.64%	75.24%	76.91
PER	89.44%	86.43%	87.91	87.78%	83.44%	85.56
Overall	85.46%	81.88%	83.63	82.06%	77.54%	79.74

Table 8.3: CoNLL03 evaluation of universal training.

to transform data with multiple dimensions into data with reduced dimensions. This technique is particularly popular for visualizing high-dimensional data on a two- or three-dimension plot. The idea of t-SNE is to minimize the difference between the two distance matrices across each point on the original space and the reduced space.

The diagrams in Figure 8.7 visualize the feature vectors to investigate how they change through the universal training and the custom training. In these diagrams, each dot represents a word. A black dot is a word that was present in the initial vocabulary set. A red dot is a word newly encountered during custom training. Figure 8.7(a) shows the randomly initialized values in the feature vectors before the pre-training. The values are mostly scattered in the two dimensional space without forming any specific patterns, with a few areas which are denser. The values in Figure 8.7(b) are generated from the feature vectors which were originally copied from the pre-trained feature vectors and then universally trained on *CoNLL_train*. Here, the dots are spread around more than in the case of the random initial vector. We believe that across the many iterations of the training the values in each word representation have moved to establish a uniform distance from one another. Meanwhile, we can observe a new pattern: a few dots form short lines. This linear pattern can be also spotted in the following two figures. Figure 8.7(c) shows the feature vectors obtained by copying the universally trained feature vectors in Figure 8.7(b) and updating them by training on *email_traina*. The feature vectors in Figure 8.7(d) are generated in the same way but trained with *email_trainb*. While these two figures maintain the forms and patterns of Figure 8.7(b), the newly introduced red dots tend to stand close to some other red dots, thus forming a group. These groups likely incorporate named-entities that never existed in the initial vocabularies and that are very relevant to a particular

	Training			Testing		
	Prec.	Recall	$F_{\beta=1}$	Prec.	Recall	$F_{\beta=1}$
LOC	83.83%	80.58%	82.17	83.00%	77.27%	80.03
MISC	79.14%	72.02%	75.41	72.31%	69.96%	71.11
ORG	85.56%	84.41%	84.98	78.08%	75.17%	76.60
PER	89.28%	86.32%	87.77	87.21%	82.90%	85.00
Overall	85.25%	81.91%	83.55	81.48%	77.41%	79.39

Table 8.4: CoNLL03 evaluation of custom training.

user, such as the name of a friend or a local restaurant.

8.6.2 NER Performance Comparison

In LN-Annote, the universal training is done with supervised learning on the CoNLL03 training (or the *dev*) dataset. Table 8.3 shows the general NER performance of universal training. In particular, the left half of the table is tested with *CoNLL03_train* and the right half is tested with *CoNLL03_testa*. The results show that in general Precision is higher than Recall for every entity type and, based on Equation 8.1, the F_1 scores remain in the range $[70, 90]$. Obviously, all the NER performance results are a bit lower on the testing dataset than on the training dataset because the parameters currently used in this experiment are originally learned from the training dataset.

Meanwhile, Table 8.4 presents the NER performance of custom training. As described in Section 8.2.3, the parameters used in this experiment are the outcome of custom training, which takes as input the learning parameters from universal training. Then, the custom training further learns from a local email dataset (*email_traina*), updating the parameters. Like for Table 8.3, the results of Table 8.4 were tested on the CoNLL03 training (left) and testing dataset (right). Therefore, comparing the tables shows how much (negative) impact, such as catastrophic forgetting, was introduced in learning by custom training.

The results show that most values remain the same, with a slight decrease compared to the values in Table 8.3. Hence, catastrophic forgetting is avoided. Note that on both the training and testing datasets the recall value of *Location* (LOC) is higher (80.58 and 77.27) than before (80.02 and 75.87). This possibly means that the custom training with the local email dataset improves the

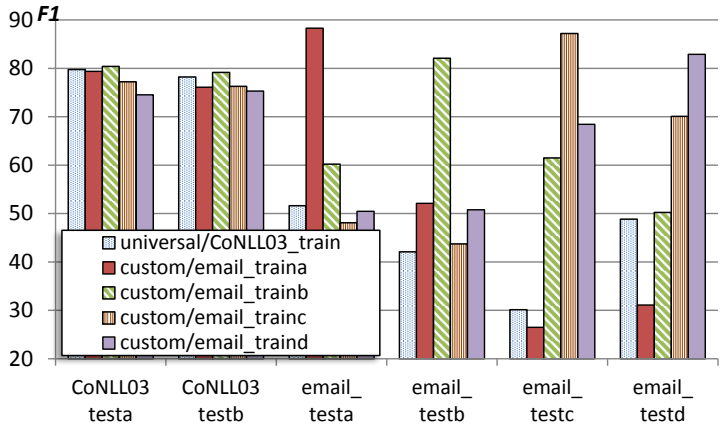


Figure 8.8: F_1 scores of universal and custom training.

parameters so that the algorithm can better recognize Location entities. For instance, there is no sentence that contains the two consecutive words "on Moscow" in the training dataset while the testing dataset has it. The NER algorithm with the universally trained parameters tagged "Moscow" in that sentence in the CoNLL03 test dataset as *O* (non-entity). On the other hand, customized parameters which learned from a local email dataset made the NER algorithm tag "Moscow" in that sentence as *I-LOC* (Location). This implies that the custom training can improve the performance not only on the specific dataset for which it is trained but also on the general dataset.

8.6.3 Cross Evaluation of Learning

In this experiment, we use five parameter sets including: the one universal-trained with *CoNLL_train* and four parameter sets custom-trained with *email_train[a-d]*. The bars in Figure 8.8 indicate evaluated values with different training and testing datasets. The evaluations are grouped by the six testing datasets: *CoNLL_test[ab]* and *email_test[a-d]*. For example, the leftmost bar indicates the F_1 score obtained from testing *CoNLL03_testa* with the parameters learned from *CoNLL03_train*. Likewise, the rightmost bar is obtained by testing *email_testd* on the parameters trained with *email_traind*. The first two groups present similar F_1 scores, ranging from 74 to 81. This shows that all these parameters can stably recognize entities in *CoNLL_train* even after custom training. In the next four groups, the customized parameters show a strong F_1 on the test datasets from the same user's emails on which the customized parameters were trained. In fact, a parameter set trained on a user's emails will never be used on a different user's emails in real systems. However, these evaluations across different custom datasets are conducted to analyze how custom training affects the perfor-

Platform	API
Model	Moto G
Chipset	Qualcomm MSM8226 Snapdragon 400
CPU	Quad-core 1.2GHz Cortex-A7 (Krait)
GPU	Adreno305 400Mhz
Main Memory	1GB
GPU Memory	441MB
Flash Storage	16GB
LCD Resolution	720 x 1280
Base OS	Linux-3.4.42 (LTS)
Platform	Android 4.4.4
OpenCL	Embedded Profile v1.1

Table 8.5: Device Specification.

mance of NER on different personal datasets. An interesting point is that the two bars, *email_testc* on *email_traind* and *email_testd* on *email_trainc*, have higher F_1 scores than other cross evaluation, e.g., *email_testb* on *email_trainc*. We speculate that this can happen when the two users share many named-entities, e.g., by having common friends or living close to one another. Similarly, the cross evaluation tends to have a certain level of correlation between the two customized parameter sets. This, however, does not mean that the relationship is always symmetric. For instance, evaluation of *email_testb* with a parameter set from *email_trainc* has a low F_1 of 43.73 while *email_testc* with *email_trainb* has 61.5, close to the average cross evaluation score.

8.6.4 OpenCL Performance Speedup

In this section, we evaluate our efforts on executing the NER training algorithm to relieve the increased CPU occupation and power consumption caused from having more computations on the mobile devices. The experiments are done on a low-end Android phone. The detailed specification of the tested phone is listed in Table 8.5.

Figure 8.9—8.11 compare the execution time of custom training by using the CPU and the GPU on the mobile device. The white bars represent the execution time of the CPU implementation written in Java. The red bars show the execution time of the GPU implementation written in

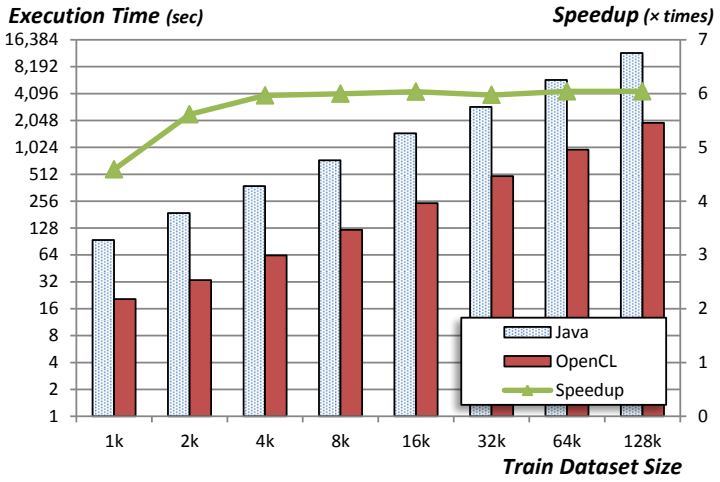


Figure 8.9: Execution time comparison (H=64).

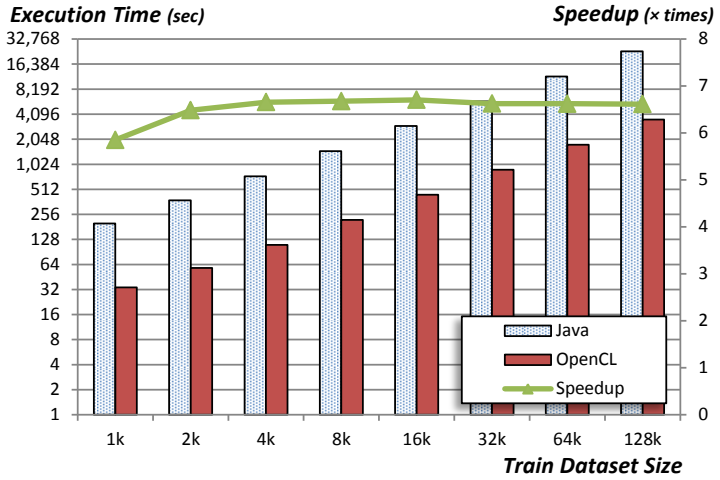


Figure 8.10: Execution time comparison (H=128).

OpenCL and embedded in the Java application through the Java Native Interface (JNI). The green curves indicate the speedup achieved by the OpenCL implementation (ran on the GPU) over the Java implementation (ran on the CPU).

Figure 8.9 shows the execution time of the training algorithm when the hidden layer size H is set to 64. The execution times of both Java and OpenCL implementations grow proportional to the input size, or the number of vocabularies in the training data set. In most cases, the speedup is close to 6, while the speedup is 4.5 and 5.5 for train dataset sized 1k and 2k, respectively. This interesting phenomenon occurs because for a small dataset the overhead from the OpenCL kernel invocation takes a large portion of the total execution time. This overhead includes times for initializing the

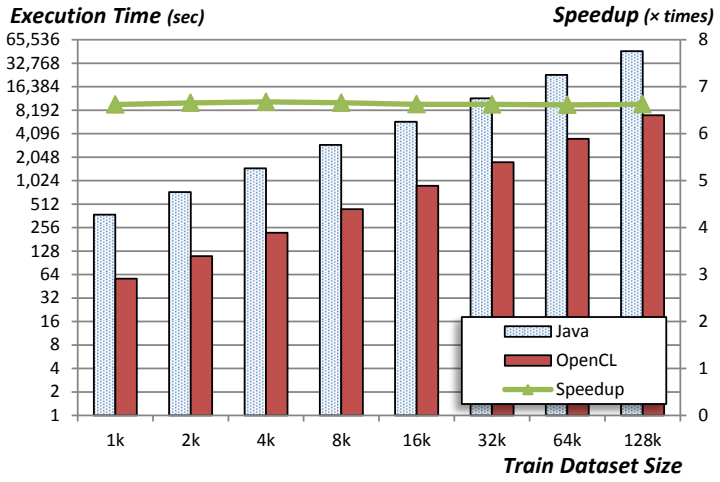


Figure 8.11: Execution time comparison (H=256).

OpenCL context, compiling the OpenCL kernel, and copying kernel arguments.

Figure 8.10 is the execution time when H is set to 128. While increasing H improves prediction performance, it takes more time to complete the computations. Compared to Figure 8.9 the execution times are almost doubled. This means that the size of the hidden layer impacts the amount of computations proportionally. The lower points on the left end of the green curve are observed also in this figure. The bending slops is more gradual than in Figure 8.9. The overall speedup from the previous figure is increased because the OpenCL performance gets better as H increases. This is because our OpenCL implementation exploits the benefit of the concurrent hardware threads, which execute the matrix operations parallelly.

Figure 8.11 presents the execution time when H is 256. Since the amount of computations required in each iteration has increased because of the larger H value, the lower speedups observed in the previous figures do not appear in these experiments. The overall speedup remains the same without a large improvement with respect to the previous experiment.

8.6.5 OpenCL Energy Saving

Our next set of experiments is about energy consumption. Since we have achieved a good speedup by utilizing the mobile GPU, we also expect to see some reduction in energy consumption. To measure the power consumption of each application and component, we used an open source software called *PowerTutor* [353] and a profiling tool, called *Trepn*, provided by the chipset vendor [257].

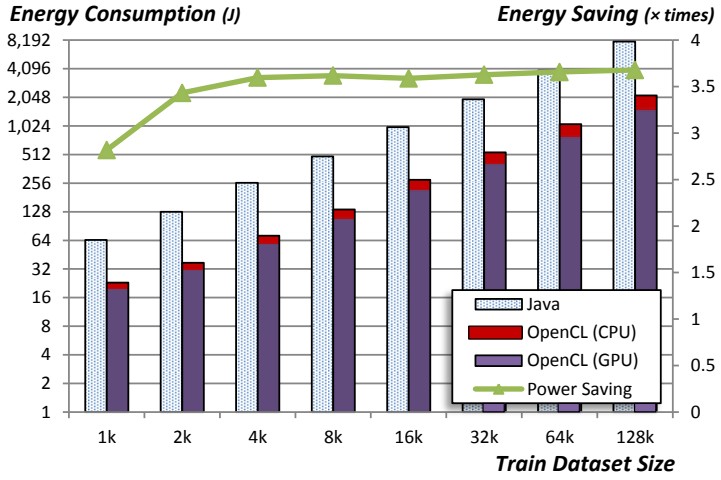


Figure 8.12: Energy consumption comparison (H=64).

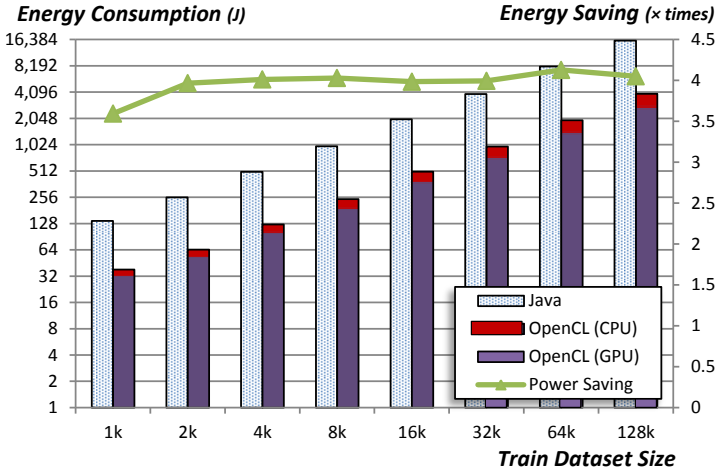


Figure 8.13: Energy consumption comparison (H=128).

Figure 8.12 shows the energy consumed by the Java implementation executed on the CPU and by the OpenCL implementation run on the GPU. The red portion of the energy consumption of the OpenCL implementation is spent by the CPU while the rest is spent by the GPU. These experiments confirm that the energy consumed on the training for the same amount of input dataset could be reduced to one fourth by using the mobile GPU. This reduced energy consumption mostly came from the decreased execution time by the GPU. In fact, the power dissipation is even higher while the algorithm runs on the GPU. For instance, when the algorithm runs on the CPU the average power consumption by the CPU is approximately 0.682 Watt. On the other hand, the average power consumptions by the GPU and the CPU when the algorithm is executed on the GPU are around

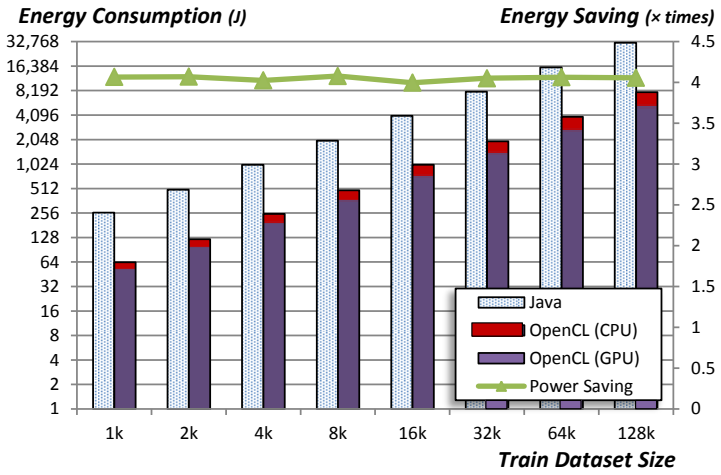


Figure 8.14: Energy consumption comparison (H=256).

1.02 Watt and 0.108 Watt, respectively. Another interesting point is that the power dissipation on the CPU, while the GPU is in use, is very low. This is because our OpenCL implementation is one consolidated kernel, thus not relying on the CPU during the entire iteration.[6] We speculate that the 0.108 Watt is mostly spent on the static power dissipation and on the maintenance of the Android app main thread, including event handling of the user interface.

Both Figure 8.12 and Figure 8.13 show lower energy savings for small datasets, i.e., 1k and 2k. Compared to the green curves in Figure 8.9 and Figure 8.10, the green curves in Figure 8.12 and Figure 8.13 stagger slightly. We presume that this is due to small errors introduced by the power measurement tools we used.

8.7 Impact on Mobile Cloud System Design

One of the advantages of our approach is to decrease the computational burden in the cloud. Although the server computers in the cloud are faster than mobile devices by at least one order of magnitude, in some cases the large number of mobile devices hides the gap in computational power. This is important for the design of MCC systems where many users access the cloud service with their own mobile devices. In this section, we discuss the design aspect of our extraction system by comparing two different approaches: 1) having the universal training and all the custom trainings on

[6]There exist some GPUs that consume CPU resources even inside the GPU kernel, although the GPU we used in the experiments does not.

the cloud and 2) having the universal training on the cloud and the custom trainings on the mobile devices.

Let's assume that our system delivers a personalized service to ten million users who access it through their mobile devices. In general, the required number of computers depends largely on the characteristics of the service. The average number of users one server computer can handle for our service could fall in a range between 1,000 and 10,000. The lower bound, 1,000 users, is calculated as follows: according to the Amazon Web Services (AWS)'s Total Cost of Ownership (TCO) calculator, the cost for using 10,000 servers (thus making each server handle 1,000 users) is $9,224,460.[7] This would take a large portion, if not most, of the revenues that a service provider with ten million service users can make.[8] On the other hand, there are various factors that limit the processing scale of a server computer to stay under 10,000 concurrent users. One factor is the challenge imposed by the network connection capacity [251; 207].

Despite this considerable number of computers, performing all the custom training tasks only on the cloud will likely incur a large delay in the service preparation cycle. For instance, the previous experiment on custom training of 128,000 words with H=128 took 23301.93 seconds on the mobile CPU but only 3523.09 seconds on the mobile GPU, as shown in Figure 8.10. The execution of the same training task on a server computer with an Intel Xeon E5-2690 CPU takes 2527.19 seconds. When each of 10,000 servers has 32 CPU cores, this will allow each CPU core to execute the custom training for 31.25 users. Suppose that we have a synchronous batch system (such as Hadoop) for the custom training where each CPU executes one custom training task for one user. Each user requests custom training everyday. The requests, however, arrive at a random time of the day. There are 32 synchronized execution time slots a day, thus making each slot be 2700 seconds. Note that this time slot is large enough for custom training on a cloud machine (2527.19 seconds) and 32 slots can sufficiently handle 31.25 users. When we assume that the user requests follow normal distribution across the 32 time slots, we get an average number of requests in each slot equal to 1.812. By applying *Little's law* [271], the average response time is derived as $1.812 * 2527.19 = 4579.27$ seconds. This is slower than the response time of 3523.09 seconds that is achieved with the training

[7]This cost includes 32 computing cores, 128 GB memory, and 16 TB storage space [6].

[8]The Average Revenue Per User (ARPU) varies across the companies, ranging from $1.21 (Facebook) to $12 (eBay) [227].

on the mobile devices, where there is no delay in response time for processing requests from other users. In other words, the training on the mobile devices may also lead to better performance. Finally, this approach will also reduce the computational burden on the cloud, thereby reducing the cloud cost.

8.8 Related Work

LN-Annote is related to some existing studies that focus on enhancing the model parameters for distinct cases. For instance, domain adaptation is an approach to transfer the feature vectors obtained on the source domain to the target domain [68]. In speech recognition, speaker adaptation is used to to improve the recognition performance by adapting the parameters of the acoustic models to better match the specific speaker's voice [274]. LN-Annote can be roughly categorized as distance supervision because the custom training uses the local contacts and SNS accounts to label the email datasets [265]. One of the characteristic differences between these approaches and LN-Annote is that LN-Annote uses information and computational capacity available on the local device. This addresses some privacy concerns and reduces the computational burden on the cloud as shown in Section 8.6 and 8.7.

8.9 Concluding Remarks

We proposed and implemented Locally customized NER-based Annotation (LN-Annote), a new method to extract personal information from emails stored locally on personal mobile devices. Our implementation is based on a newly designed neural network model that works well for the two main phases of learning that characterize LN-Annote: universal training (performed in the cloud) and custom training & testing (performed on the mobile devices). We also developed two methods for optimizing the training of the neural network: one is based on the use of feature templates and the other on leveraging the GPUs that are present on the mobile devices. The experimental results show the feasibility and effectiveness of LN-Annote. In particular, they demonstrate how the use of custom trained parameters actually improves the performance of NER on the local email data without reducing its performance on the dataset used for universal training.

Chapter 9

Query-Division Reverse Offloading: a Ranking Model for ASR

Audio Stream Retrieval (ASR) is an emerging class of applications in the area of audio retrieval. ASR clients periodically query an audio database with an audio segment taken from the input audio stream to keep track of the original content sources in the stream or to compare two differently edited audio streams. Together with Jaehwan Koo, Karl Stratos, and Luca Carloni, I recently developed a series of ASR applications such as broadcast monitoring systems, automatic caption fetching systems, and automatic media edit tracking systems. In these automated systems, ranking is particularly important because the systems take the most likely result (top-ranked) for their purposes.

In this chapter, we propose a probabilistic ranking model that utilizes the distance between the query and the results along with two ASR-specific observations: 1) the transitions from one content source to another content source within the input stream have a probabilistic pattern across the content sources and 2) the two adjunct queries from a stream are likely from the same content source. Meanwhile, we also introduce a query optimization technique and a simpler version of the ranking model for the systems where the technique is applied. In order to train and test the model, we create a new set of audio streams and make it publicly available. Using these open audio streams as well as commercial audio streams, we evaluate the proposed model with an extensive set of experiments. Our experimental results confirm that the proposed ranking model effectively sorts the retrieved results, thus reducing the errors when used in various applications. Finally, We show

how we used NETSHIP to develop the client-side search and rank app for Android and to measure the performance difference brought by the optimization technique.

9.1 Introduction

Audio segment retrieval, i.e., searching information about the original audio content with a query of an audio segment, is a technology that is increasingly used in the area of audio retrieval [103; 323]. This can be viewed as a special type of information retrieval, called *Query by Example* (QbE), which searches results identical or similar to the example provided in the query from the user instead of searching with the constraints or keyword terms in the query [356; 303]. Likewise, in audio segment retrieval the user's query contains an audio segment as an example and then the retrieval server returns information about identical or similar audio sources. One of the most widely used applications of audio segment retrieval hitherto is music identification [204; 14; 326; 70]. This application takes a segment of music to search information on the original music from the database server. While the input query in audio segment retrieval is mostly a piece of audio, a typical output result can be comprised of a combination of the following: 1) metadata such as the creator, the date of creation, the ID, or the title of the content [53]; 2) an access to the whole content (or similar content [58]); 3) derivative works like a subtitle, a caption, or a lyric file [101]; and 4) the relation between the input segment and the original content, i.e., the position of the input segment in the searched content [138].

We present a new class of audio retrieval applications that we developed. In these applicaations, which are already used in production by tens of broadcast stations worldwide [16], the action of audio retrieval is repeatedly performed online with queries taken from an audio stream. First, we developed a realtime broadcast monitoring system. In this system, the client periodically (e.g., every second) queries the database with an audio segment taken from an on-air video channel and logs the response that contains information on which content sources have been broadcasted. This system is used by broadcast stations that want to monitor the stability of the actual broadcast and by advertisers who want to count how many times their commercials are exposed on air. The second type of application is an automatic caption-fetching system. In this system, while the user watches a video stream, the client video player extracts an audio segment from the video, queries the ID and

the position of the video currently played, and fetches the corresponding caption from the database to display it on the screen along with the video. This system significantly simplifies the process of caption matching and editing. For instance, the caption is automatically displayed no matter if the video stream is edited (scene cutting or speed adjusting) or the video content that has been played in the stream ends and a new one starts, as long as the audio segment of the current video content is found in the database.

We call this new class of applications *Audio Stream Retrieval* (ASR) as it has certain characteristics distinct from traditional audio retrieval. In ASR the result usually consists of a pair of items: a content ID and the position of the audio in the query within the retrieved result. For example, the result can be ('Hotel California', 105000ms) when the query is made from the part of the song that goes "Welcome to the Hotel California". This allows ASR applications to track the sequential progression of the input stream. Second, it consists of multiple, periodic, and online retrieval actions over the sequential audio stream. More importantly, the results of audio stream retrieval are used as part of automated systems in many applications. For example, the automatic caption-fetching system uses the highest ranked result for caption matching. Performing this requires the application of a very effective ranking model on top of the retrieved results. It also needs a high *recall*, which ensures that almost always the relevant result from the database is included in the result set. Another characteristic is that in many cases there is a probabilistic relation between the previous content source and the next one.

In this chapter, we propose a probabilistic ranking model using these characteristics in addition to the similarity of the results and the query, which has been used also in traditional audio retrieval. Particularly, our contributions include:

- reviewing fingerprint-based ASR systems,
- developing a probabilistic ranking model for ASR,
- inventing an optimization technique for ASR systems,
- creating and opening a public audio stream suite for training and testing, and
- evaluating the proposed ranking model with an extensive set of experiments.

In the next section, we provide background information on fingerprint-based ASR systems. In

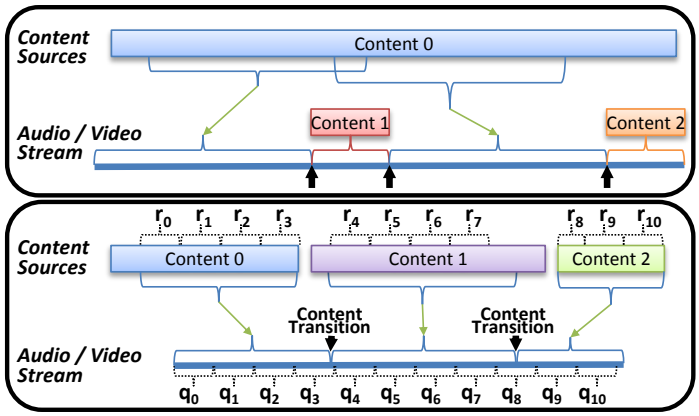


Figure 9.1: Two examples of audio streams: an edited stream for a TV channel with commercials (top) and a stream with a user occasionally changing the content source (bottom).

Section 9.3 we present the probabilistic ranking model for ASR and show how we train it. Next, we introduce an optimization technique that reduces the number of queries in ASR, thus helping the scalability of the retrieval servers, and an appropriately modified ranking model (Section 9.4). In Section 9.5, we show how we developed and optimized the ASR system using NETSHIP. Last, in Section 9.6 we describe the used datasets and evaluate the ranking model.

9.2 Audio Stream Retrieval

In this section, we explain some concepts and background information necessary to understand the development of the ranking model for ASR.

9.2.1 Audio Streams

A *content source* or simply *content* is the original audio or video file from which the fingerprints for database is generated. An audio or video *stream* is a sequence of excerpts from various content sources. ASR clients create fingerprints from the streams, combine them into a query with the fingerprints, and send the query to the retrieval server.

Figure 9.1 shows two examples on how streams are made of multiple contents. The figure presents also queries q_k and the ideal results r_k corresponding to q_k. The top example is a typical stream that can be observed on TV channels. During a TV show (Content 0) some commercials

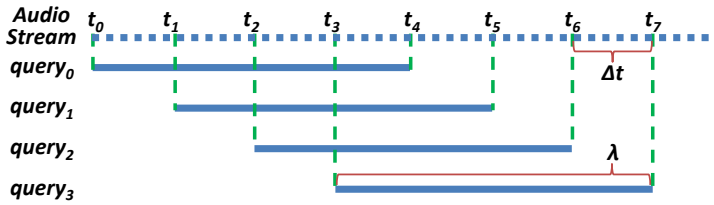


Figure 9.2: Buffer excerption.

(Content 1 and 2) are played intermittently in the middle of the show. The other example presents a user who watches different content sources back-to-back. The user may watch the content from the beginning (or any arbitrary position) to the end (or any other arbitrary position). Across these two examples, the goal of ASR is to find out the original content sources and the positions from which the excerpts came.

Another important concept is *content transition* or simply *transition*. A transition is a change of the content source in a stream. In Fig. 9.1 the transitions of the streams are indicated as black arrows denoting changes of content sources.

9.2.2 Retrieving Audio Fingerprint

Our proposed ranking model is built for audio stream retrieval systems that store and retrieve audio data using *audio fingerprints*. The audio fingerprints, or simply fingerprints, are designed to represent certain amount of audio data. The fingerprints are represented with data types that have a limited size, usually much smaller than the original audio data. Fingerprint creation is a non reversible transformation; restoration of the original audio from the fingerprints is impossible. The design of fingerprint is robust when the same contents constantly produces (almost) the same fingerprints regardless of the different encoding methods used for each content source, the volume of the content, or even some (unintended) noise. Likewise, distinct audio sources are required to produce distinct fingerprints. There are a number of well-known features that can be extracted from an audio source to be used in the fingerprints [224; 60]. The fingerprints in our systems are created from the original audio by splitting it into several overlapping frames and applying the Short-time Fourier transform on them [47; 176]. The values are grouped by the audio frequency ranges and then composed into a multidimensional vector, used as a fingerprint to represent the original audio frame.

The size of a fingerprint can be determined by the length of the total audio content sources hosted in the retrieval system and the desired level of possible duplication of two fingerprints from different audio frames. The fingerprint duplication probability can be modeled as in the case of the Birthday Problem and efficiently computed using the Taylor Series approximation method [110]:

$$p(n, d) \approx 1 - e^{\frac{-n(n-1)}{2d}} \tag{9.1}$$

where n is the number of total fingerprints from the existing content sources and d is the number of all possible fingerprints allowed in the given multidimensional vector space.

For example, in one of our production systems we wanted to keep the probability of fingerprint duplication under 0.2% while supporting 3 distinct trillion fingerprints.[1] Hence, we used a 2^{96} possible fingerprint space, which can be encoded in a 12-bytes value and gives a 0.16% duplication possibility, and we mapped the audio frequency values into 12 dimensions with one byte for each.

9.2.3 The Query

In ASR, the client makes a query after every *query interval* $\Delta t = t_{k+1} - t_k$, as shown in Figure 9.2. The k-th query takes an audio segment from t_k to t_{k+1} of the audio stream. The time notation does not necessarily mean realtime but the content time. The query interval is realtime in the automatic caption-fetching system because it also plays the video or audio. Instead, in the automatic media-edit tracking system that compares two audio streams without playing them, the client sends another query as soon as the results from the previous query return.

The *query length* λ of a query is the length of the audio segment used to make the query as presented in Figure 9.2. The query length is an independent concept from the query interval. In Fig. 9.2, for example, the query length is $4\Delta t$. A longer query length may allow the retrieval server to pick up more precise results out of a large number of similar content sources, but it also may cause confusion for the same server if the content in the input stream changes very frequently (shorter than the query length). Meanwhile, a shorter query interval may lead to a faster discovery of content changes but it can burden the retrieval server with too many requests, thus limiting its

[1] This amount is equal to all the video contents YouTube currently hosts and is based on the number of fingerprints produced, e.g., ≈5.38 in our case, for each second of audio.

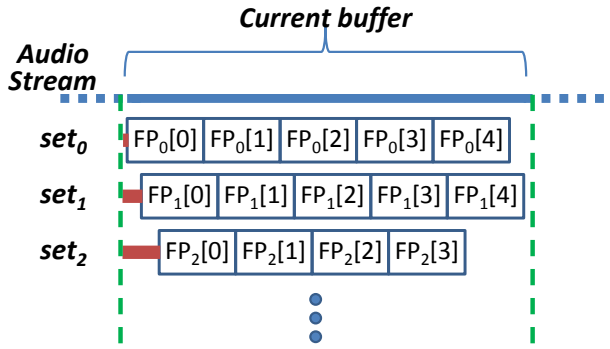


Figure 9.3: Overlapping Fingerprint Sets in a Query.

scalability. Therefore, the independence of these two concepts, query length and query interval, allows more flexibility of configuration in the ASR systems.

Some of our fingerprint systems use a concept called *overlaps*. Figure 9.3 shows how a query consists of multiple overlapping fingerprint sets with different offsets represented as a thick red bar in front of each set. The search engine chooses one of these overlapping sets based on the distance of the fingerprints between the set and the results. This chosen set is then used for the rest of the process. In order to keep the necessary (memory) resources low, these overlaps exist only in the query, not the database.

Even if it is excerpted from one audio stream, an ASR query may contain more than one content source. In other words, a query can be created from a segment that includes one or more content transitions. Consequently, ASR systems have to handle this type of queries.

9.2.4 The ASR Workflow

Figure 9.4 shows the flowchart of the proposed audio stream retrieval. It is an iterative process in which the client and the retrieval server interact repeatedly. The client decodes the audio or video content stream, extracts an audio buffer, and generates fingerprints. Then, it creates a query that includes the generated fingerprints. If it is not the very first query in the stream and the client has previous results, then these previous results are also included in the query to be used in the retrieval process, e.g., result ranking.

There are two ways to use information from the previous retrieval results. First, the server can maintain the previous results in a session-based retrieval system. Second, the client can keep the

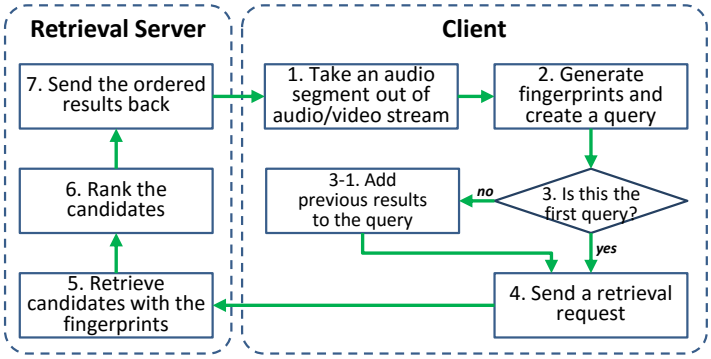


Figure 9.4: Flowchart of the proposed ASR.

previous results and send a query with the previous results in a session-less system. In our products, we use the latter type for the design of queries as illustrated in the flowchart.

The server searches its database with the last fingerprint in the query and gives a ranking score for each retrieved result. These results are sent back to the client. Many text-based information retrieval systems use the ranking model not only to rank the documents but also to retrieve documents from the database. Differently from these text-based retrieval systems, the ASR systems use separate models for retrieval and ranking. In this chapter we focus on the ranking of the ASR systems.

9.3 The Proposed Ranking Model

Ranking in information retrieval is a way to provide the user with the relevance of documents regarding the required information expressed as a query. It plays a significant role in information retrieval as it suggests the most relevant documents. It is often the case that the users cannot possibly check all the retrieved documents or they want to see the most relevant document [161]. Ranking the results is important in audio stream retrieval for the same reason. In addition, the results of ASR are used as a part of automated systems in many applications.[2] For example, the broadcast monitoring system uses only the highest ranked result and the automatic caption-display system employs a client-side heuristic-based module that picks one result from the top-k ones.

[2]There are also a few applications, e.g., a copyright monitoring system, where the users want to identify all the streams that use a certain portion of their work. In this case, high recall is more important than ranking.

One approach used in ranking audio retrieval can be used also as a basic method to rank audio stream retrieval results: i.e., to use the distance between the fingerprints in the query and the fingerprints in the retrieved content sources. This allows high recall and precision thanks to the design of fingerprints as described in Section 9.2.2. The rank achieved by the distance works well as long as there are no content sources that have similar portions. If there exist, however, some content sources that have similar portions, then it is important to distinguish the results because their distances to the query will be too short to be excluded. The more relevant result should be the actual content and position of the audio the user is feeding whose truth value, however, cannot be known. The less relevant results should be the content sources and the positions of audio segments that are different from the truth value but have a similar portion to what the user is feeding. This is a problem almost impossible to solve only with fingerprint distances because there are content sources that contain precisely identical audio parts, e.g., a same song (or a sound effect) used in two different movies.

Wherever the truth values of the relevance property of each result is unknown to an information system, we assume that the information available for ranking in the system is at best probabilistic. This is motivated precisely by the Probability Ranking Principle [161] that states:

> `If retrieved documents are ordered by decreasing probability of relevance on the data available, then the system's effectiveness is the best that can be obtained for the data.`

Next, we present a probabilistic ranking model for audio stream retrieval based on the given information in the database and the query. In audio stream retrieval, we can probabilistically anticipate the current results based on the knowledge of the previous results. The analysis of our applications in production makes us observe the presence of a probabilistic relation that characterizes the transition between content sources. This is due to the following reasons: 1) today's video content hosting services provide their users with recommendations which often lead to a high correlation in the transitions between similar videos; 2) the users have a particular interest that may affect their watch patterns, e.g., watching all the soccer games a team has played during a league or watching a sequence of episodes of a TV show; and 3) a publicly available video stream (or channel) is watched by many users, i.e., many users will watch the same sequence of video content sources such as TV commercials and shows, which the video stream (or channel) contains.

9.3.1 Notation

In this section, we recall and extend some important notation from the Probabilistic Relevance Framework [266] for the development of an ASR ranking model.

First, we assume that the property of relevance between the query and the result is binary. The relevance status is represented as either rel (relevant) or $\overline{\text{rel}}$ (not relevant).

Two monotonic functions f_1 and f_2 are *equivalent as ranking functions* when the ranked orders of any results by the two functions are identical:

$$\text{rank equivalence: } \propto_q \text{ e.g., } f_1() \propto_q f_2()$$

A result **r** in audio stream retrieval is a pair of values: a content ID c and a position in the content p. A query thus corresponds to a content source c from p to $p + \lambda$, where λ is the query length,

$$\mathbf{r} = (c, p) \text{ where } c \in C, \text{ and } 0 \leq p.$$

C is a set of IDs of all the possible content sources in the system. Querying the database returns a set **R** of results ordered by their ranking scores. As mentioned in Section 9.2.4, an audio stream retrieval query comprises also the results of the last query. Hence, we define a query **q** as a combination of a list of fingerprints and a previous top-ranked result. We denote the k-th query as:

$$\mathbf{q}_k = (\{\text{FP[i]} \text{ — } 0 \leq i < n \text{ and FP}[i] \in F\}, \mathbf{R}_{k-1}.r_0)$$

where $\mathbf{R}_k$ denotes the retrieved results for q_k and F denotes the set of all possible fingerprints in the feature space. The feature space is an imaginary 12-dimensional space where each axis corresponds to each feature value in the fingerprint. In our previous example of 12 bytes fingerprints, there are 2^{96} possible fingerprints in F.

9.3.2 Ranking Model Development

In this section we derive a probabilistic ranking model for ASR. Instead of using the cosine similarity between tf-idf[3] vectors as in vector space models for text-based information retrieval [269], we estimate the relevance between a query and a result using three factors: ① the history (or the

[3]term +frequency-inverse document frequency.

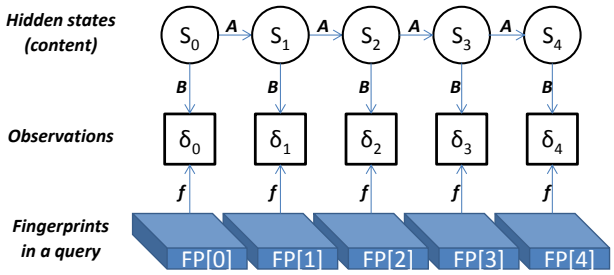


Figure 9.5: Content Probability based on HMM.

previous result) of how the content sources in the input stream are changing (Section 9.3.2.1); ② the distance between the fingerprints in the query and the ones the result indicates (Section 9.3.2.2); and ③ the heuristics that in many retrieval cases the duration of the excerpted content portions in the streams is usually longer than the query interval (Section 9.3.2.3). In particular, ② is what can be used also for audio retrieval whereas ① and ③ are information available only in ASR.

We evaluate each result $\mathbf{r}$ in $\mathbf{R}_k$ retrieved from the database with the last fingerprint in the query $\mathbf{q}_k$ as shown in Figure 9.4. The relevance of a result $\mathbf{r}$, a pair of content c and position p to a query $\mathbf{q}$ can be written as $P(\mathrm{rel}|c,p,\mathbf{q})$. We use the relevance probability as the ranking score so that the results are sorted by this score in descending order. We transform the relevance probability with the following steps:

$$P(\mathrm{rel}|c,p,\mathbf{q}) \propto_{\mathbf{q}} \frac{P(\mathrm{rel}|c,p,\mathbf{q})}{P(\overline{\mathrm{rel}}|c,p,\mathbf{q})} \tag{9.2}$$

$$= \frac{P(c,p|\mathrm{rel},\mathbf{q})}{P(c,p|\overline{\mathrm{rel}},\mathbf{q})} \frac{P(\mathrm{rel}|\mathbf{q})}{P(\overline{\mathrm{rel}}|\mathbf{q})} \tag{9.3}$$

$$\propto_{\mathbf{q}} \frac{P(c,p|\mathrm{rel},\mathbf{q})}{P(c,p|\overline{\mathrm{rel}},\mathbf{q})} \tag{9.4}$$

$$= \frac{P(c|\mathrm{rel},\mathbf{q}) \cdot P(p|\mathrm{rel},c,\mathbf{q})}{P(c,p|\overline{\mathrm{rel}},\mathbf{q})} \tag{9.5}$$

First, the relevance probability is transformed by odds-ratio in Equation (9.2). We get Equation (9.3) by applying Bayesian inversion. In Equation (9.4) we drop the terms that are not related to the results (c and p), while preserving the ranking. Finally, the probabilities of content and the position are split in Equation (9.5). The remaining three terms are expanded sequentially in the following sections.

9.3.2.1 Content Probability

The term $P(c|\text{rel}, \mathbf{q})$ is the probability of c given the query $\mathbf{q}$ and a relevance rel of a result $\mathbf{r}$ (c, p) with respect to $\mathbf{q}$. To estimate this probability we apply the Hidden Markov Model (HMM) [49] to ASR by treating the content sources as the hidden states and the fingerprints as the observable outputs. Figure 9.5 illustrates this more in detail. Using this model, we can compute the probability that c is the current content given the fingerprint sequence in the query. We assume that the probability of a change between content sources is affected only by the current content, not the previous ones. This allows us to use HMM for estimating the content probability.

On top of HMM, we use the Forward algorithm [296] to calculate a 'belief state', the probability of a state at a certain time, given the history of observations. For instance, in Figure 9.5, we can get the probability of each content source in the retrieved results being the current content c in the stream, i.e., $S_4 = c$. Thus, the result of the Forward algorithm for c on the last hidden state is the probability of c being the current content from which the last fingerprint FP[4] has been generated.

One of the advantages of HMM is that the observation and state sequence length can vary. The number of fingerprints in the query can also vary due to the different offsets and the frequencies of the query interval and the audio stream. For instance, one query may have 40 fingerprints while the next query has 42 fingerprints, although their buffer lengths are set to be the same. Thus, HMM is a good method to compute the content probability using the fingerprint sequence.

There are two unknown probabilities sets in HMM. The state transition probability $A = P(c_i|c_{i-1})$ denotes the likelihood that the content is changed from c_{i-1} to c_i. The emission probability $B = P(\delta_i|c_i)$ denotes the likelihood that δ_i can be observed when the content is c_i. In ASR, A implies the probabilities of transition between content sources and B implies the probabilities of a fingerprint being observed at a state. These HMM probability parameters are learned as described later in Section 9.3.3.

If the size of the states and the observation space are large the amount of computation required for training the HMM parameters becomes high. For this reason, we reduce the number of states and the observation space in our production systems, e.g., by clustering the content sources and picking the top-k content sources. Also, since the fingerprint space F is too large to efficiently compute the parameters, we reduce it so that it maps into the smaller observation space by using a dimension-reduction algorithm such as Vector Quantization [125]. An observation value δ is obtained from a

fingerprint by a dimension-reduction function f:

$$\delta_i = f(FP[i]). \tag{9.6}$$

where i is the index of the fingerprints in the query.

Thus, for a given query $\mathbf{q}$, the probability that content c is relevant to $\mathbf{q}$ is calculated by the Forward algorithm which computes the probability $\alpha_t(x)$ that the t-th hidden state is x:

$$\begin{aligned} P(c_i|\text{rel}, q) &\propto_{\mathbf{q}} \alpha_i(c_i) \\ &= P(c_i, \delta_{1:i}) \\ &= \sum_{c_{i-1}} P(\delta_i|c_i, c_{i-1}, \delta_{1:i-1}) \cdot P(c_i|c_{i-1}, \delta_{1:i-1}) \\ &\qquad \times P(c_{i-1}, \delta_{1:i-1}) \\ &= P(\delta_i|c_i) \sum_{c_{i-1}} P(c_i|c_{i-1}) \cdot \alpha_{i-1}(c_{i-1}) \end{aligned} \tag{9.7}$$

where c_i is the content, or the hidden state, from which the i-th fingerprint in the query has been created.

Therefore, we get an expanded ranking formula:

$$P(\text{rel}|c, p, \mathbf{q}) \propto_q \frac{\alpha(c) \cdot P(p|\text{rel}, c, \mathbf{q})}{P(c, p|\overline{\text{rel}}, \mathbf{q})} \tag{9.8}$$

by replacing the content probability with $\alpha(c)$, the probability that the last hidden state is the content c given the query $\mathbf{q}$.

9.3.2.2 Result Probability

The probability of the position p, given the content c, the query $\mathbf{q}$, and the relevance rel between the result and the query can be computed by the distance of the fingerprints. In other words, if the distance between the two fingerprints from the result and the query is big, it is unlikely that the result is relevant to the query. While the fingerprint sequence in the query is known, the fingerprint sequence from the database is uncertain. This is because the results are retrieved with only the last fingerprint in the query. The other fingerprints in the query could possibly come from different content sources if there were transitions within the query buffer.

There are four different approaches to decide which content source should be used in the comparison against the query to compute the distance between them.

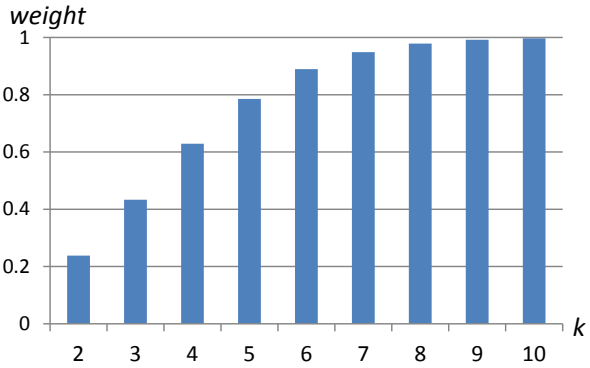


Figure 9.6: Credibility weights as function of k.

1. Using the Viterbi algorithm to determine the most likely sequence of the states (content sources) [104]. This approach, however, gives only the information on the content sources, not the positions. Thus, this requires additional efforts to decide the positions from each content source to compare with the query.

2. Retrieving results with not only the last fingerprint but also all other fingerprints in the query. This gets the most likely sequence of content sources and positions. This could incur, however, too many retrieval requests to the database server, therefore lowering its scalability.

3. Taking a fingerprint sequence linearly from the database by assuming that the sequence in the query is from one continuous content source. This assumption can lead to a miscalculation in the distance value, when the content is changed in the query.

4. Using a hybrid approach of 2 and 3. This assumes that the fingerprints in the query are from the same content until one of the fingerprint distances between the query and the content exceeds a certain threshold.

We used the hybrid approach to obtain a fingerprint sequence for each result retrieved by the last fingerprint from the query and match them against the query.

The average distance between two fingerprint sequences X and Y over the k last fingerprints is:

$$d(X, Y, k) = \frac{\sum_{i=n-k-1}^{n-1} ||X_i - Y_i||}{k} \tag{9.9}$$

where $||x - y||$ denotes the Euclidean distance between two fingerprints x and y and $k \in [2, n)$ is

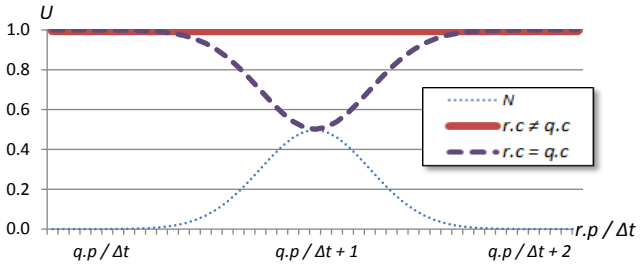


Figure 9.7: $\mathbb{U}$ and a normal distribution N.

the number of compared fingerprints in the sequence. A weighted and inverted distance is:

$$d^I(X,Y) = \max_k \left\{ \frac{1}{1+d(X,Y,k)} \cdot \frac{\Gamma(\lfloor k+1 \rfloor, 4)}{\lfloor k \rfloor !} \right\} \tag{9.10}$$

where $\frac{\Gamma(\lfloor k+1 \rfloor,4)}{\lfloor k \rfloor !}$ is the weight term based on a cumulative distribution function of the Poisson distribution, as shown in Figure 9.6. This function gives more credibility to the distance values obtained from longer sequences while it reaches a plateau at a certain length of the sequences. For instance, the credibility is only 0.238 for a length of the sequences equal to 2 while 0.949 for a length 7. As the length increases beyond 7, the return value slowly grows up to 1.

The probability of the position p given c, rel, and $\mathbf{q}$ from the distance function is:

$$P(p|\text{rel}, c, \mathbf{q}) \propto_q d^I(\mathbf{q}.\text{FP}, \text{FP}_\mathbf{r}) \tag{9.11}$$

where $\mathbf{q}$.FP is the fingerprint sequence in the query and $\text{FP}_\mathbf{r}$ is the sequence obtained with the last fingerprint in the result $\mathbf{r}$ using the hybrid approach.

The expanded ranking formula is:

$$P(\text{rel}|c, p, \mathbf{q}) \propto_q \frac{\alpha(c) \cdot d^I(\mathbf{q}.\text{FP}, \text{FP}_\mathbf{r})}{P(c, p|\overline{\text{rel}}, \mathbf{q})} \tag{9.12}$$

9.3.2.3 Irrelevant Result Probability

Let $\mathbf{q}_{k-1}$ and $\mathbf{q}_k$ be two adjacent queries with a query interval Δt and r_{k-1} be the top-ranked result retrieved by $\mathbf{q}_{k-1}$. If one result r_k retrieved by $\mathbf{q}_k$ has the same content as r_{k-1} and its position is Δt behind the position of r_{k-1}, then, it is likely that there were no content transitions between the two queries in the stream and therefore this result is less likely to be irrelevant. For example, if $\text{R}_k.\mathbf{r}_0 = (0, 2000)$ and $\text{R}_{k+1}.\mathbf{r}_0 = (0, 3000)$ while the predefined source interval time Δt is 1000,

we know that $\mathrm{R}_{k+1}.\mathbf{r}_0$ would be the correct result with a high probability. Then, these results $\mathrm{R}_k.\mathbf{r}_0$ and $\mathrm{R}_{k+1}.\mathbf{r}_0$ are said to be *contiguous*. We use this heuristic information to model the probability U of a result when the query and the result are irrelevant for a given $\mathbf{q}$.

$$
\begin{aligned}
U &= \begin{cases} 0, & \text{if } r.c = q.c \text{ and } q.p + \Delta t = r.p \\ \frac{1}{|R|}, & \text{otherwise} \end{cases} \\
&\propto_q (U + \frac{1}{|R|}) \cdot \frac{|R|}{2} \\
&= \begin{cases} 0.5, & \text{if } r.c = q.c \text{ and } q.p + \Delta t = r.p \\ 1.0, & \text{otherwise} \end{cases}
\end{aligned}
\tag{9.13}
$$

Due to the various different sampling and fingerprinting intervals, the position within the content always contains some small errors, thus making the comparison impractical. As shown in Figure 9.7, we revise this function with a probabilistic normal distribution function:

$$
\mathbb{U} \approx 1 - [r.c = q.c] \cdot 1.25 \cdot \mathcal{N}(x, \mu, \sigma^2) \tag{9.14}
$$

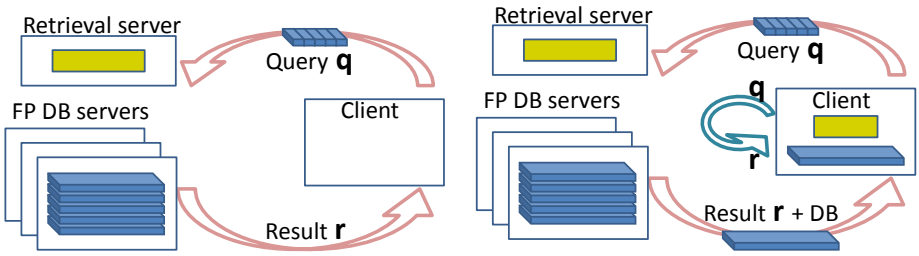
where [P] is an Iverson bracket which returns 1 if P is true, $x = 4\frac{r.p}{\Delta t}$, $\mu = 4\frac{q.p}{\Delta t}$, and $\sigma^2 = 1.0$. This results in the final ranking formula:

$$
P(\mathrm{rel}|c, p, \mathbf{q}) \propto_q \frac{\alpha(c) \cdot d^I(\mathbf{q}.\mathrm{FP}, \mathrm{FP}_\mathbf{r})}{\mathbb{U}} \tag{9.15}
$$

This ranking model is used to calculate a ranking score for each result. Then the retrieval server sorts the results with their ranking scores in descending order.

9.3.3 Learning Parameter Values

The two HMM parameters we used in the proposed ranking model are one of the key factors to make it successful. The ideal values for these parameters are different for various applications. Even with the same applications, different composition of content sources in the database or different user-transition patterns observed in the input streams require different parameter values for the best ranking performance. For this reason, many recent ranking systems learn the parameter values used in the ranking model to achieve more appropriate parameter values, thus satisfying its users better. This is called *Learning to Rank* [209]. By applying machine learning algorithms the ranking model can work more closely to the user patterns.



(a) Non-optimized Querying (b) Optimized Querying

Figure 9.8: Querying optimization.

When applying machine learning, it is difficult under certain circumstances to obtain the training dataset for the learning model. We collect training datasets from the ASR applications that include the input (a query) and labeled output (a result) for the ranking model. We extracted from the ranking model datasets the input (dimension-reduced fingerprints and the previous content) and output (a content source) to train and test the embedded HMM. This allows us to train the HMM parameters separately from the ranking model. We obtain the initial emission probabilities from the fingerprint database for each content source by counting the frequencies of fingerprint observations. This accelerates the learning of the emission probabilities based on the actual occurrence of fingerprints within the input stream.

We use two different methods to learn the HMM parameters for systems under different circumstances, particularly regarding the availability of the label (truth value) of the current content in the stream for training. First, in the systems where the label of the content in the given training stream is available (supervised training), we calculate the transition probabilities and the emission probabilities online by using the content label and the observations generated from the fingerprints in the query. Second, in the system where the content label is unavailable (unsupervised training), we use the Baum-Welch algorithm to learn the HMM parameters. The Baum-Welch algorithm is based on an expectation-maximization algorithm to find the maximum likelihood estimate of the parameters of a HMM given a set of observed feature vectors [254].

9.4 Optimized Querying and Ranking

We invented an optimization technique for ASR to reduce the number of queries. This technique relies on two facts. First, the input content stream is from a continuous source for a certain amount of time that, in many cases, exceeds the interval of queries. Second, the fingerprint from one source content is stored serially, e.g., as a file, in the database server (before being loaded into the main memory for retrieval using a specific data structure such as trees). Hence, *data locality* can be used in ASR for reducing the number of queries from the clients, thus allowing a better scalability to the servers.

Figure 9.8(a) illustrates the iterative process of ASR without optimization. The client queries the retrieval server. The server has various features such as query processing, retrieval from database, and ranking, collectively represented as a yellow box. This process can be improved by letting the client have a certain amount of fingerprints from a continuous content source and query itself. In Figure 9.8(b), the client has a 'light' version of search functions (a yellow box). First, the client sends a query to the server. When the retrieval server finds some results for the query, it returns to the client a fingerprint sequence file along with the results. After a query interval, the client queries its own local search engine instead of querying the server. It can decide if the next result is still in the local database by comparing the score of the top-ranked result from the local search to a configurable threshold ϵ. If a good result is found within the local database, it simply returns it. Otherwise, it forwards the query to the server and lets the server retrieve the queried fingerprint from the entire database. This approach significantly reduces the number of queries made to the servers.

While the ranking model remains unchanged in the server after the optimization technique is applied, another ranking model still has to be implemented on the client. This is because searching from the local database can return multiple results due to the presence of similar fingerprints at different positions within the content. The ranking model on the client is simpler than the server's ranking model. The reason is that the local search cares only for the cases without content transitions. If a content transition is suspected in the stream because there is no result whose distance to the query is less than ϵ, then the local search stops and forwards the query to the server. Therefore, the client side ranking model can assume that all the subsequential results have the same content. Specifically, the content probability and the Iverson bracket from the irrelevant result probability

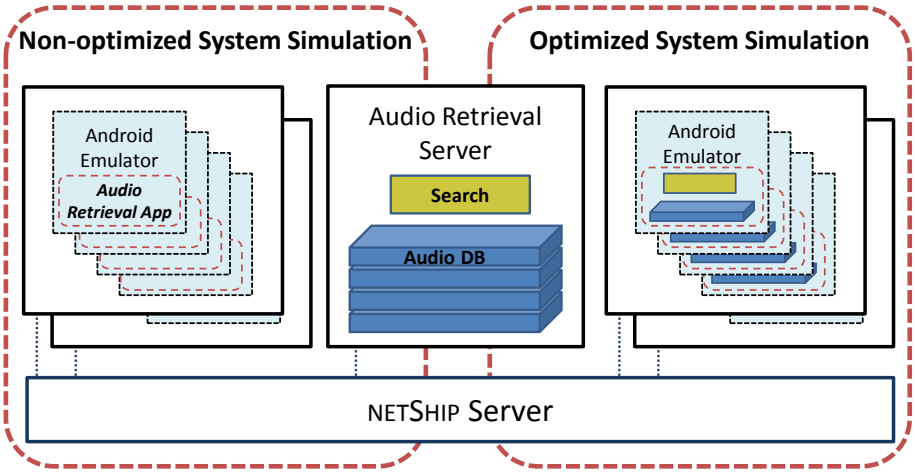


Figure 9.9: Simulating the audio retrieval system using NETSHIP.

are removed, thus leaving the client-side ranking model as:

$$\frac{d^I(\mathbf{q}.\text{FP}, \text{FP}_\mathbf{r})}{1 - 1.25\mathcal{N}(x, \mu, \sigma^2)} \tag{9.16}$$

This client-side ranking model is used to sort the results retrieved from the client search engine. It is used also in the comparison to the threshold to decide when the client needs to stop local searching and forward the query to the server.

9.5 Designing the ASR system on NETSHIP

The ranking system for fingerprint-based audio stream retrieval is a new type of MCC system, where the mobile devices send queries to the server periodically. Hence, the communication between the server and the mobile clients is very frequent, in many cases resulting in delays and insufficient performance. The proposed optimization technique moves a contiguous portion of the database to the mobile client and searches locally until the audio source in the input stream on the client changes.

Figure 9.9 shows the simulation of the ranking system. The set of configurations on the left corresponds to virtual machine instances used in the non-optimized system simulation. Each of these virtual machines executes up to four Android emulators. The Android emulators run the audio retrieval app which send audio retrieval queries to the audio retrieval server. The audio retrieval

server is a virtual machine instance which has the search and rank capabilities while storing the audio fingerprint database. This audio retrieval server instance is shared with the two simulation configurations: the non-optimized system simulation and the optimized system simulation. Alternatively, the set of configurations on the right corresponds to virtual machine instances used in the optimized system simulation. The Android app executed in the optimized simulation contains the local search and rank capability along with a portion of the database from which an audio chunk can be searched.

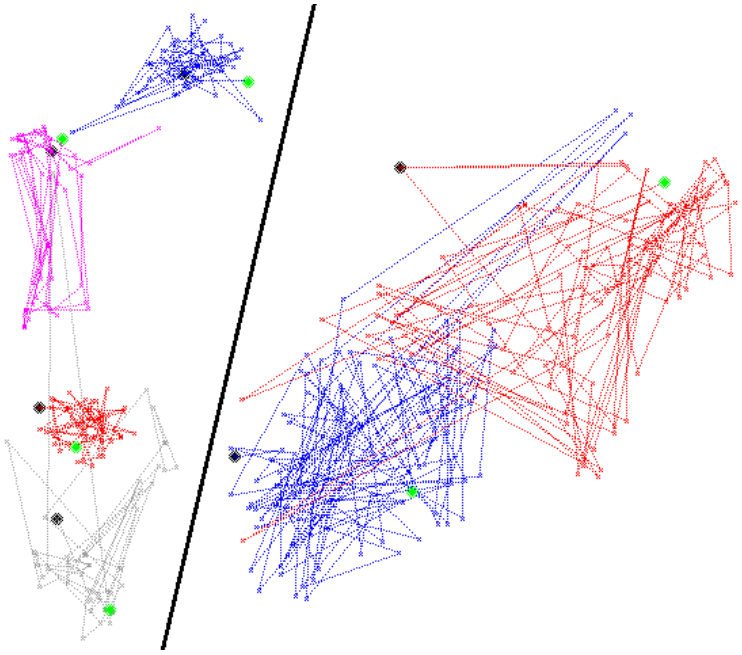
We use NETSHIP to develop the client-side search and rank app for Android and to measure the performance difference brought by the optimization technique. Particularly, we are able to achieve the following experimental results using NETSHIP:

- The scalability of the designed system including the maximum number of clients a server instance can support.
- The execution time speedup achieved by the optimization technique.
- The best possible values of the threshold ϵ for the lower power consumption and the faster execution speed.
- The power consumption estimates of each implementation.

9.6 Experiments

For the evaluation of our ranking model, we created a set of open audio streams [4] that present distinct characteristics in terms of content transitions. Along with the open streams, we also evaluate our model with a dataset from our commercial database. This commercial dataset is, however, inaccessible for the public. The streams consist of excerpts from some audio field recording archives named freefield1010 [295]. In the streams we created, transitions are biased in the sense that the transitions from a content source tend to converge to a few content sources rather than being evenly spread across many content sources. The stream group names *bias0* and *bias100* mean unbiased (completely random) streams and completely biased streams, respectively. More detailed information is available on the website.

[4] Our *Open Audio Stream* suite is available at *http://openaudiostream.org*



(a) Freefield1010 (b) Music and Voice

Figure 9.10: Fingerprint stream visualization.

The experiments on the Open Audio Stream discussed from Section 9.6.2 to 9.6.4 are tested with the server that has loaded all the fingerprints created from the 7,690 freefield1010 wave files. The production dataset used in the same experiments includes over 300,000 hours of audio with an average length of approximately 1.12 hours. We denote each term of the ranking model as follows: D for the distance-based result probability, H for the history-based (using previous results) irrelevant result probability, and C for the HMM-based content probability. Particularly, we denote the combinations of these terms as D+H and D+H+C to evaluate the additional impact of each term as opposed to D as the baseline method.

9.6.1 Fingerprint Stream Visualization

A well-designed fingerprinting method is an essential prerequisite for an effective ASR system. If the fingerprints for different content sources are not very distinct from each other, the ranking model based on the fingerprint distance cannot work accurately. To demonstrate how our fingerprinting method works for distinct audio sources, we visually compare fingerprints from different audio streams. We used an algorithm called t-SNE, which reduces multidimensional values into lower

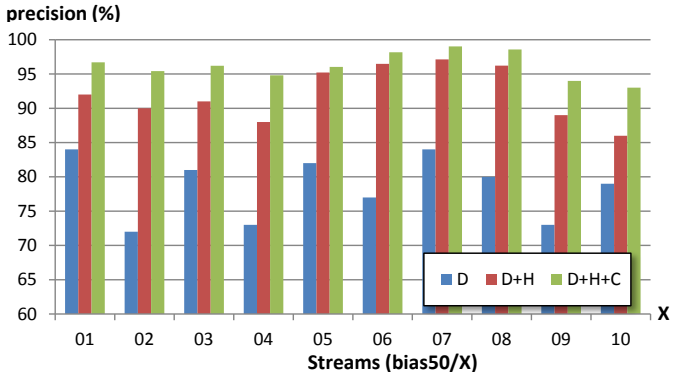


Figure 9.11: Precision of top-ranked result (Open Audio Stream).

dimensions, e.g., 12 to 2 [321]. This is the first approach to visualize audio fingerprints with the t-SNE algorithm.

In Figure 9.10 we mark each fingerprint in an audio stream with a small cross and connect them with lines in the sequence as they appear in the stream. Therefore, each separate contiguous line represents a separate audio stream, starting with a black dot and ending with a green dot. Figure 9.10(a) shows four different audio samples taken from: a shallow stream [102071] (blue), a waterfall [102089] (red), chirping birds [102724] (purple), and a male voice [102864] (gray), where [ID] is the ID of the audio source taken from freefield1010. Figure 9.10(b) compares two sounds from a production database: the song *Gangnam Style* (blue) and the first spoken Wikipedia page called *Interstate 15 in Arizona* (red). We can see that sounds that come from the same source, and therefore are similar to each other, yield similar fingerprints, thus drawn closely in the figure. Contrastingly, different sounds from different content sources are positioned further apart. This confirms that this fingerprint system can be effectively used for distance-based ranking in audio stream retrieval.

9.6.2 Ranking Model Evaluation

In this section we evaluate the precision of our ASR system expressed as the average percent of correct top-ranked results out of the total number of queries. We included ten sets of streams from the *bias50* directory of the open audio stream (shown in Figure 9.11) and eight sets of streams from one of our production databases (shown in Figure 9.12). For each set of streams we ranked the retrieved results with D (the distance-based result term only), D+H (D and the irrelevant result

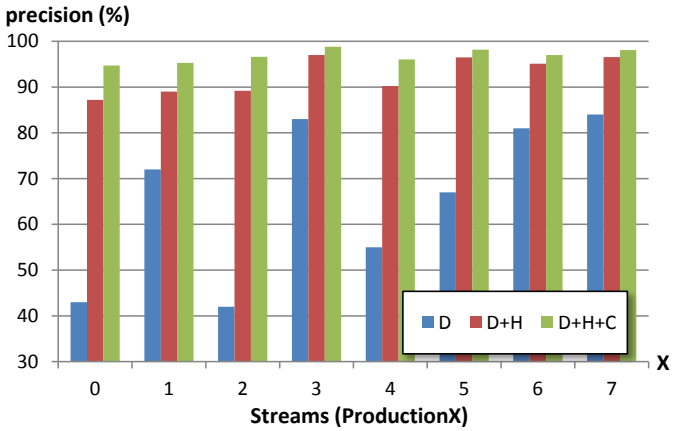


Figure 9.12: Precision of top-ranked result (Production Stream).

term using previous results), and D+H+C (D, H, and the content term). In each set of streams, the more elaborate rank model gives more precise ranking results. Particularly, H improves the precision from 10% points to more than 30% points. The introduction of H has more impact on the production database due to the longer average excerpt duration of each content source within the streams, i.e., 34 minutes vs. 5.7 seconds. Thanks to the additional improvement brought by C, the complete ranking model (D+H+C) results in over 95% of precision across all eight evaluated streams. In Figure 9.12, the precision obtained by D is quite low (below 50%). This is because in production there are many factors that make the audio fingerprints vary from the precisely desired value such as: significant speed modulations, transcoding with a high loss rate, or noises recorded in the audio. Hence, the ranking based only on the distance is inappropriate to be used in production.

9.6.3 Evaluation Breakdown By Examples

By delving into some examples gathered during the experiments in Section 9.6.2, this section demonstrates how the precision improvement presented in the previous section is actually achieved. Table 9.1 shows two queries from the stream *biased10/07/0019*. The portion of the stream from which the two queries are taken is originally made from content source with id *110917* and positions from 7800 ms to 7900 ms and from 7900 ms to 8000ms, respectively. Retrieving with these queries and ranking should bring the results with the same content and positions as top-ranked ones. For instance, with the D ranking model, the first query (110917@7800) brings two results where the top-ranked one is (110917@7811), which is correct (because the system allows positions within

Query		D			D+H		
c	p	c	p	Score	c	p	Score
110917	7800	**110917**	**7811**	**0.34**	**110917**	**7811**	**0.75**
		110917	3014	0.07	110917	3014	0.07
110917	7900	*169249*	*9230*	*0.19*	**110917**	**7904**	**0.33**
		110917	**7904**	**0.17**	*169249*	*9230*	*0.19*
		168490	3014	0.15	168490	3014	0.15

Table 9.1: Ranking Score Example: D vs. DH.

Query		D+H			D+H+C		
c	p	c	p	Score	c	p	Score
170470	9200	**170470**	**9230**	**0.49**	**170470**	**9230**	**0.374**
		159171	3955	0.34	159171	3955	0.012
17077	600	*42200*	*7911*	*0.35*	**17077**	**598**	**0.054**
		162452	*5462*	*0.33*	*42200*	*7911*	*0.011*
		17077	**598**	**0.32**	*162452*	*5462*	*0.009*
		69965	3955	0.15	69965	3955	0.004

Table 9.2: Ranking Score Example: DH vs. DHC.

$\pm$ 250 ms in this example). The second query, however, brings three results where the supposedly-correct one is ranked second. This incorrect behavior may happen whenever there are very similar parts in the content database. The D+H model, on the other hand, ranks (110917@7904) as the top for the second query while keeping the rank of the first query also correct. As the two queries are adjacent, having one query interval Δt = 1 second in-between, the irrelevant result term boosts the score of the result of the second query up to 2 times. This is one of the most powerful factors of the ranking model as points that can benefit from this score boosting are very frequently observed in the user streams.

Table 9.2 shows two queries from the stream *bias60/03/0012* and their retrieval results ranked by two models: D+H and D+H+C. The portion of the stream from which the two queries are taken is composed of two different sources: from 9200 ms to 10000 ms for *170470* and from 400 ms to

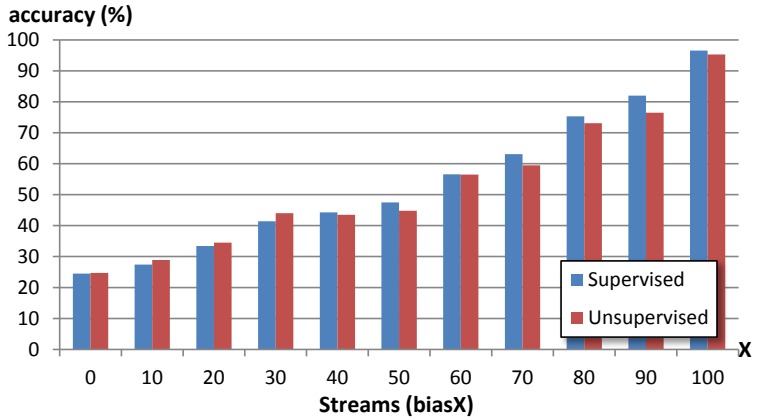


Figure 9.13: Content Prediction Accuracy (Open Audio Stream).

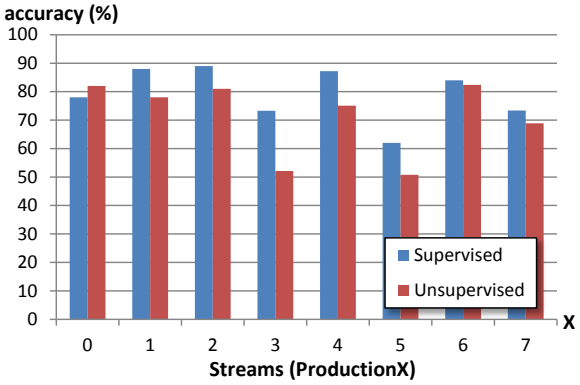


Figure 9.14: Content Prediction Accuracy (Production Stream).

1400 ms for *17077*. In other words, there is a transition in this part of the stream, changing from *170470* to *17077*. The dataset we used for training and testing the ranking model, particularly the HMM-based content probability, is *bias60* of the open audio stream. With D+H, the first query gets a correct top-ranked result (170470@9230). The second query, however, results in the correct result (17077@598) being ranked as third. This also happens when there are similar sounds in other content sources. The D+H+C model, on the other hand, fixes this problem by multiplying the score with the probability of the content source *17077* being the content of the second query. This is computed with the HMM Forward algorithm using parameters trained on the *bias60* streams and the fingerprints in the query.

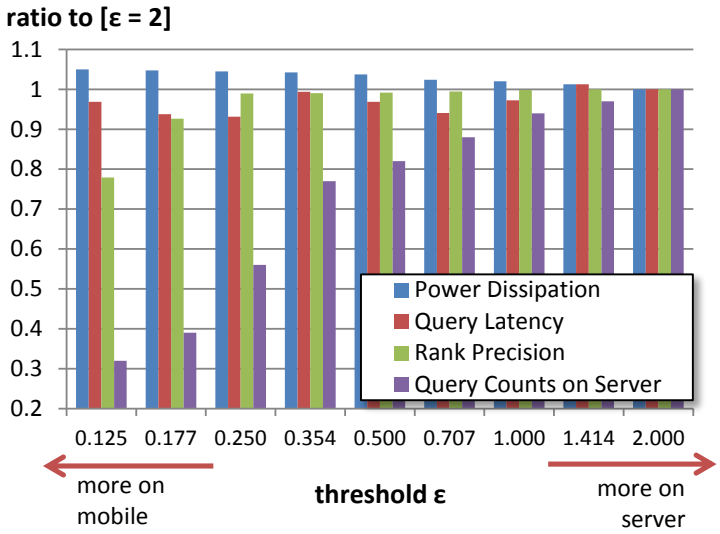


Figure 9.15: Evaluating Optimization on Mobile.

9.6.4 HMM Evaluation

This section shows the accuracy of content prediction, calculated by the number of correctly estimated most-likely content sources. Figure 9.13 and 9.14 show the values obtained with sets of streams from the open audio streams and a production dataset, respectively. The ranking model is trained with the ten streams randomly selected from each set and is tested against all the streams in the group. Some interesting observations can be made. First, in the case of the open audio stream datasets (*bias30* - *bias90*), the accuracy increases as the streams are more biased. This is because the state-transition probabilities learned from some streams in a dataset are likely to work well with other streams in the same dataset if the transitions in that dataset are more biased. Second, comparing the accuracy measurements shows that the production dataset has a bias level similar to some sets of the open audio streams, ranging from *bias50* to *bias90*. In many cases supervised training results in a better performance by around 5% points to 20% points.

9.6.5 Optimization Configuration

In these experiments we show how we can configure the threshold ϵ introduced for the optimization technique in Section 9.4 to achieve different effects on the various aspects, e.g., mobile energy budget, cloud costs, or network latencies. The value of the threshold ϵ can vary from zero (usually non-inclusive) to two (inclusive). If ϵ is zero, every query except the first one is processed on the

client. If ϵ is two, the optimization technique will never be used and every query is made to the server. As ϵ decreases, fewer queries are sent to the server and more queries are processed on the client. The user of this ranking model may want to find the sweet spot that maximizes some particular aspects. For instance, ϵ can be set between one and two whenever the rank precision is the only important requirement. If the system is struggling under the deficient resources on the server while a slight rank precision decrease is permitted, then setting ϵ to a value between 0.25 to 0.5 can be a good choice.

Figure 9.15 shows the experimental results when the optimization technique is applied to an ASR client running on a mobile device. The specification of the tested device is described in Table 9.3. In order to measure the power dissipation of each configuration during the executions of the client application, we used open source software called *PowerTutor* [354] and a profiling tool, called *Trepn*, provided by the chipset vendor [257]. The power dissipation increases as more computational resources are needed with a smaller ϵ. This increase, however, is kept under 5% by limiting the use of network resources, e.g., the Wi-Fi module or the TCP/IP stack running on the CPU, required to query the server. The query latency remains somewhat comparable across the different ϵ values. This phenomenon along with the slight variations on the measured values is due to the network, which at times incurs randomly varying latencies. The observed rank precision remains at a comparable level until we reduce ϵ from 2 to 0.25. As we further decrease ϵ the rank precision starts to drop.

It is interesting to notice that performing the search on the mobile device has a fairly negligible impact on both execution latency and power dissipation. Since the in-memory database is less loaded, the search task on the mobile device requires a much smaller computation than executing the same task on the server. For example, while the mobile database may contain 200k fingerprints, the server database may be loaded with 60 million fingerprints.

9.7 Concluding Remarks

We introduced audio stream retrieval that has distinct characteristics such as periodic querying, content excerpt length usually longer than the query interval, and the content transitions in the input streams. For ASR, we proposed a probabilistic ranking model that uses a distance of fingerprints

Platform	API
Model	Moto G
Chipset	Qualcomm MSM8226 Snapdragon 400
CPU	Quad-core 1.2GHz Cortex-A7 (Krait)
Main Memory	1GB
Flash Storage	16GB
LCD Resolution	720 x 1280
Network	Wi-Fi 802.11 n
Base OS	Linux-3.4.42 (LTS)
Platform	Android 4.4.4

Table 9.3: Device Specification.

between the query and the retrieved results. The ranking model uses ASR-specific information to improve the ranking results. We also introduced an optimization technique that can reduce the number of queries made to the servers. We developed a suite of audio streams for training and testing purposes that we made publicly available online. The experimental results show that our ranking model achieves high precision and that the proposed optimization technique obtains 40% reduction of queries sent to the servers without affecting the precision (and with only 5% more power dissipation in the case when the client is a mobile device).

Part III

Conclusions

Chapter 10

Co-Development Process of MCC Applications and Tools

As the size of software projects is increasingly growing and the complexity of the developed software is also rising [56], an increasing number of software development methods have been adopted. Particularly, the iterative development process is a popular method based on a cyclic procedure for software designs [40]. It is known for cost-efficiency [223], ease of prediction [32], and effectiveness regardless the size of development team [279]. As shown in Figure 10.1, each step of the iterative process is repeated in sequence as the software is developed and improved. It consists of five steps:

- **Requirement Gathering.** In this step, the requirements of the desired software are extracted and investigated. While customers probably believe that they already know what they want the software to do, in many cases it still requires skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements.
- **Analysis & Design.** This is a step where the given problems are analyzed to be solved and the algorithms, tools, software architectures, techniques, and interfaces to be used are decided. Particularly, this step includes prototyping which gives the guidelines to the developers who will implement the software and to the users who will use the software product.
- **Implementation.** In this step, the design turns to writing the actual software code which actually executes following the desired behavior. In each iteration of the implementation step,

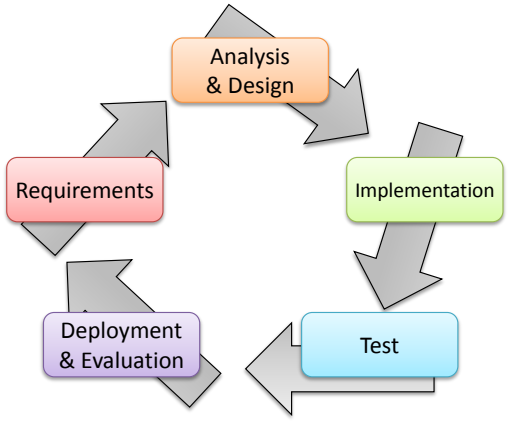


Figure 10.1: The iterative process of a software development project.

a subset of the whole software requirements, i.e., the portion of the requirements gathered in the requirement gathering and analysis step in the same iteration, is implemented. The whole software is iteratively enhanced until the full system is implemented. At each iteration, the software design is modified and the corresponding functions are added.

- **Test.** Software testing is the task of finding possible bugs in the software by executing or validating the software code. Testing is becoming increasingly important in the software development process, as the variety of the requirements grows and the complexity of the software implementation is increasing. In testing, there are two types: static testing (e.g., code reviewing, walkthrough, or inspection) and dynamic testing (e.g., unit testing, integration testing, or system testing).

- **Deployment & Evaluation.** Deployment step is where the software becomes available to its users. Generally, deployment involves software release, installation, activation, update, and version tracking. After software deployment, users can use and evaluate the software. The evaluation of the software is a significant piece of the software development process as it contributes in the form of feedbacks to determining the software requirements for the next iteration.

Each individual project presented in this thesis across Chapter 4 to 9 is developed under an iterative development process. The design tools for MCC (Chapter 4 to 6) and the MCC applications (Chapter 7 to 9) are alternatively developed in a way where one project helps another advance.

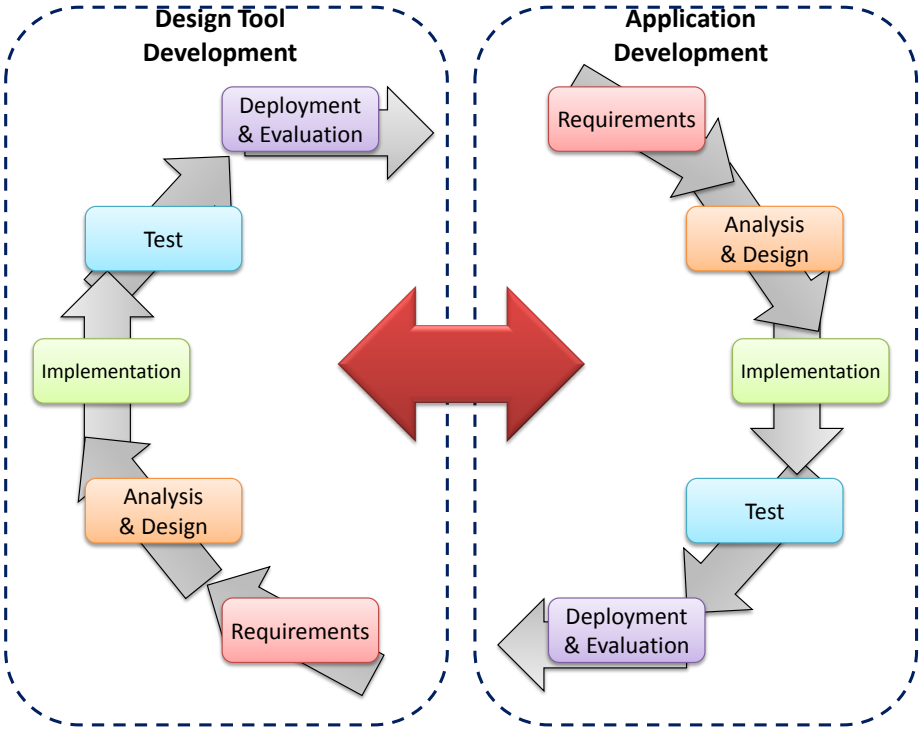


Figure 10.2: The combined, iterative development process of design tools and applications.

Figure 10.2 shows the combined parallel process of two software development projects, the development of a design tool and an application. This iterative development process progresses step-by-step inside one development project, i.e., either in the left or right side of the figure, as done in the single iterative software development project. However, two projects in each side of the combined development process can concurrently progress, contributing to one another. For example, a deployed tool can be used for the analysis & design step by offering a target system prototype and information that can be used for the new application design. The tool can also provide a virtualized environment on which the application can be implemented and tested. The users of the tool, i.e., the application designers and developers, will provide some error/bug-reports and feedback information during each single step in the application development process using the tool. This information from using the tool augments the set of requirements to be implemented during the tool development process. Meanwhile, while developing the tool, an already developed, deployed application can be used as a test case for reliability, back-compatibility, and scalability.

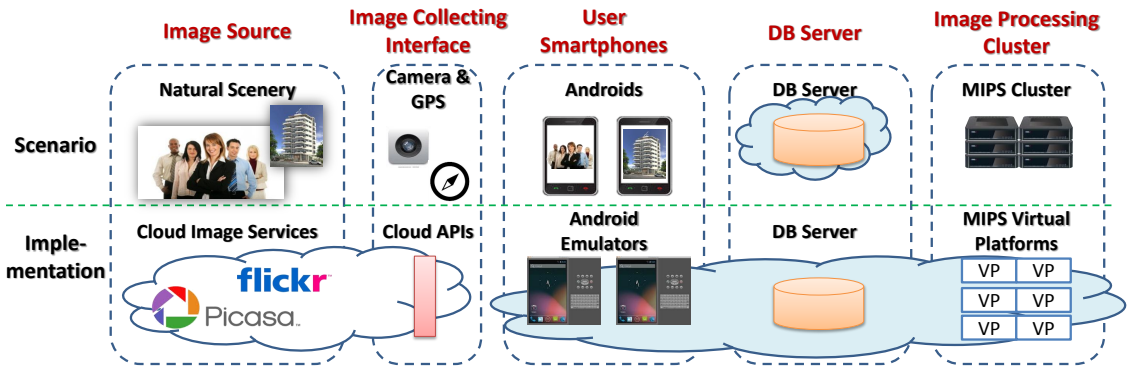


Figure 10.3: The design of the crowd estimation system using NETSHIP.

In this chapter, I show how the two lines of software projects I developed for this discussion actually helped one another across the different steps in the iterative development process.

10.1 Tool-Based Design and Development

One of the strategies to reliably develop software that is increasingly getting complex is using a good tool that supports the design and development of software. Many software development tools have been released for software architects and developers [106; 308; 188; 63]. However, as the computing systems keep evolving and new types of computing systems reach the market, the system designers and developers need new types of system design tools. Through Chapter 4 to 6 I introduced a new class of design tools that support heterogeneous computing nodes, i.e., cloud computers, embedded devices, and sensor motes. These tools also offer large-scalability that can cover the emerging MCC systems up to a very large scale, with hundreds of millions of devices. The NETSHIP simulation environment presented in Chapter 4 targets the design and testing of MCC systems that execute applications which can access cloud services. In particular, it simplifies performance and scalability analysis by making it possible to simulate the execution of the actual applications and software stacks onto virtual models of the hardware and the network.

Figure 10.3 shows an example of an MCC application introduced as Case Study II in Chapter 4 from a design tool's perspective. This system consists of smartphones, a cluster of embedded devices, and a cloud-based server. In this scenario, smartphone users take some pictures (Image Source) with geolocation information using the camera module (Image Collecting Interface) on the

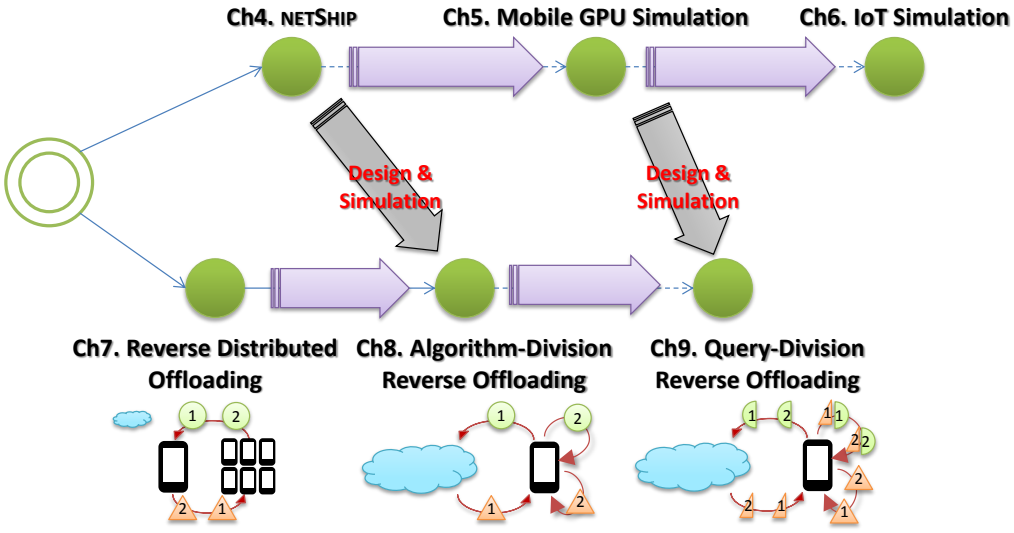
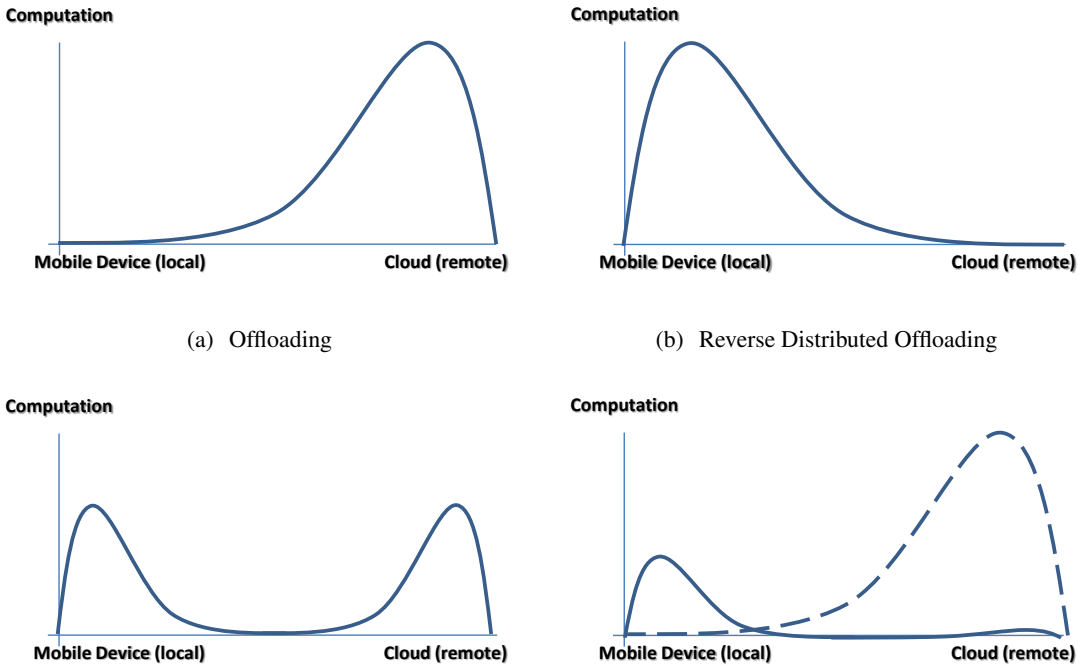


Figure 10.4: Developing MCC applications by using design tools.

phone (User Smartphones) and then upload these pictures to the database server in the cloud. There is a cluster of MIPS processors (Image Processing Cluster) that downloads the pictures from the DB server, runs an image processing application to count the number of people in the picture, and adds up the number to the geolocational sum in the DB server. In the implementation, several parts of the design are replaced by virtual counterparts. For instance, instead of physically deploying multiple Android phones, I used Android Emulators running an application that simulates the behavior of smartphone users. Due to the lack of the camera module in the emulator, images downloaded from cloud image services, such as Picasa and Flickr, through their public cloud APIs served as the user-taken pictures. Lastly, OVP MIPS instances form a cluster to run an image processing application. The Android Emulators and the OVP instances in Figure 10.3 are virtual platforms. Using NETSHIP, I built a networked VP that simulates the designed system. Given the application requirements, I used NETSHIP to gain insights on the amount of resources required for real-time processing of the pictures taken by a large crowd in a particular geographic area, Manhattan. This case study application represents an interesting example where the simulation-based design tool can be used for both the development of the application and the design of the system, allowing the observation of the system optimization, scalability, and deployment.

Likewise, I also developed the MCC applications introduced in Chapter 7 to 9 using the design tools I developed and introduced in Chapter 4 to 6. The relationship between the MCC applications



(a) Offloading

(b) Reverse Distributed Offloading

(c) Algorithm-Division Reverse Offloading

(d) Query-Division Reverse Offloading

Figure 10.5: Distinct distributions of computation in MCC systems.

and the design tools used for them is shown in Figure 10.4. The design tools were used to: develop the application programs that run on the smart devices, design the overall system architectures, and measure the performance improvement introduced by the optimization techniques. In this section, I explain how the tools have contributed to the design and development of two MCC applications: the LN-Annote system and the Audio Retrieval system.

10.1.1 Distributions of Computations

While developing the MCC applications by using the design tools, I performed design space exploration of the applications and systems we developed, inventing new optimization techniques that can increase the execution performance, decrease the cloud costs, or reduce the energy consumption. In particular, I focus on the distribution of computation across mobile devices and cloud since it fundamentally affects the design of MCC systems including system architectures, implementations, and financial aspects of system designs, i.e., the costs of mobile devices, cloud instances, and networks.

While the details of measured values in performance and energy consumption is offered in Chap-

ter 7 to 9, they are not directly comparable due to the different application programs, measurement units, platforms. Instead, in this section, I show an abstracted projection of distributions of computation in each MCC system. Figure 10.5 presents four main types of distribution of computation in MCC systems, varied by adoption of an offloading or reverse offloading technique. Figure 10.5(a) corresponds to an MCC system where the computational offloading technique is applied. In this figure, the computation is concentrated on the cloud because the mobile devices have offloaded most of the computation they need to the cloud, thus leaving little work to be processed locally. Figure 10.5(b) shows the amount of computation in an MCC system with Reverse Distributed Offloading. In this system, the computational task originated from a mobile device is distributed across multiple mobile devices instead of offloading it to the cloud. As a result, the computation is concentrated only on the mobile devices. Figure 10.5(c) presents the distribution of computation achieved by Algorithm-Division Reverse Offloading. Since the first half of the computation needed by the main algorithm is executed on the cloud and the rest runs on the mobile devices, the amount of computation is evenly split into the two sides. Last, Figure 10.5(d) shows the distribution of computation obtained by Query-Division Reverse Offloading. Since it optimizes a subset of the queries, there are two lines: the solid line shows the computation in the optimized case while the dashed line presents the case when the query cannot be optimized. In the non-optimization case, the graphs is identical to Figure 10.5(a) because the mobile sends the query to the server, which is offloading. Meanwhile, the optimization method proposed in Chapter 9 reduces the amount of computation needed to be retrieved due to the smaller size of the database from which the query is retrieved. Thus, the amount of computation processed in the mobile device is smaller than the offloaded case.

10.2 Application-Driven Development of Design Tools

Not only a design and simulation tool can play a significant role in the development of new software and hardware systems, but also the development of the design tool is largely driven by the new systems developed using the tools. Once a new tool is developed and released to its users, the users will evaluate it and give feedback. Without this information, it would be impossible for the tool developers to know which features need to be added or improved.

Figure 10.6 shows how the MCC applications helped finding the necessary features for the

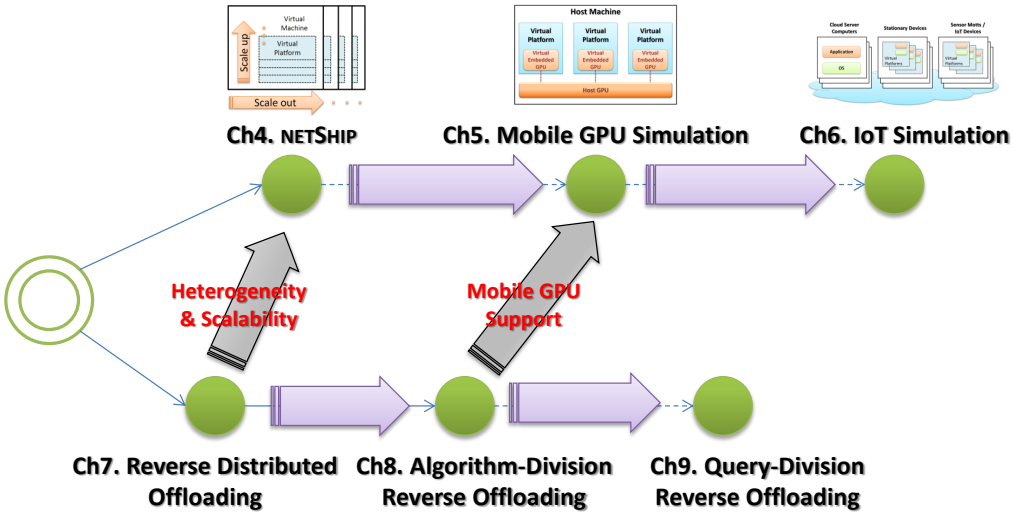


Figure 10.6: Improving design tools according to the needs from the applications.

design tool, NETSHIP. The broadband embedded computing system where reverse distributed offloading is applied is a heterogeneous and large-scale system as introduced in Chapter 7. This system consists of blade server computers as well as multiple embedded devices. I expect that the number of the embedded devices in this system can grow over millions in the future. Developing this system, I invented a novel method to port a huge-sized software framework into an embedded environment, most parts of which are automated for rapid software migration into the embedded devices. Nonetheless, there are a few points that slow down the overall development process such as deploying the ported software onto the physical devices, debugging the software across the multiple, distributed devices, and managing the physical devices (e.g., manually turning on/off the devices). These problems could be avoided by using a simulator that supports such a system, which however did not exist. Motivated by this point, I started developing the system simulator introduced in Chapter 4 that supports heterogeneity and scalability as required by the previous developed application. The requirements for the design tool development project that were derived from the broadband embedded computing system project are as:

- **Scalability.** The system could scale up over millions of devices. The scalability of the designed tool has to be large and efficient.

- **Heterogeneity.** The target system is comprised of different types of components. In par-

ticular, the network and the computing nodes that form the target system can significantly vary.

The next application project that induced an additional set of requirements for design tools was the LN-Annote system introduced in Chapter 8. Since the optimization technique used in the system, algorithm-division reverse offloading, leverages the mobile GPU present in each mobile devices, the design tool needs to support the simulation of mobile GPUs for accurate execution time measurement and efficient power-consumption estimation. The mobile GPU simulation project described in Chapter 5 was built to enhance NETSHIP by providing efficient simulation of the MCC systems where the mobile GPUs are used. Particularly, the requirements for the mobile GPU simulation project include:

- **GPU Functionalities.** The designed simulation system should be able to execute the GPU kernels without modifying the target code.
- **Execution Time Estimation.** The developed simulation tool is required to provide the estimated execution time of the target kernel.
- **Power Consumption Estimation.** The developed simulation tool is required to provide the power consumed on the mobile GPU to execute the target kernel.

Chapter 11

Future of MCC Systems and Design Tools

MCC systems have gained an increasing popularity in the market. The combination of ubiquitous mobile devices and powerful cloud computers has shown various advantages over traditional computing service architectures. As a result, there is an increasing number of emerging MCC systems and their users. Meanwhile, the efforts to design the new MCC systems in a more efficient and optimized way has led to the invention of new design tools optimized for them. It is still a beginning stage where only a small number of tools have been developed. However, both the MCC systems and their design tools are expected to grow rapidly in the near future. In this chapter, I present my vision for the future of the MCC systems and design tools.

11.1 Future MCC Systems

Due to the wide spread of smart devices, the importance of scalability in the MCC systems has been rapidly growing. A popular MCC service has a usage spike with hundreds of thousands of users accessing the service concurrently, thus relying heavily on the implemented system's scalability [180; 215; 118]. The heterogeneity of the MCC systems has become an important issue as well. The diversity in available choices in mobile manufacturers, mobile network service providers, and mobile software platforms have made today's mobile devices vastly heterogeneous [292; 198].

In future MCC systems, both scalability and heterogeneity are expected to keep increasing. A

variety of new computing devices and sensors enter our lives every day: second screens [116], IoT sensors [287], smart cars [277], and robots [84]. These new types of computing nodes will be extending the scope and boundaries of MCC systems, thus demanding higher scalability and widening their heterogeneity. Cloud-Robotics, for instance, is a new form of computing systems where robots leverage the better computational power and storage capability of cloud back-end servers [261; 177].

In this section, I describe two possible directions for the future of MCC systems. The first is Internet-of-Things, a fast-growing research area. The second is Altocumulus, a proposed intermediate cloud system.

11.1.1 Internet-of-Things

IoT systems are very similar to MCC systems in the sense that the data collected from the IoT devices is process in the central cloud. In the near future, the IoT systems can either be a part of or absorb MCC systems. The two types of systems might be merged into a newly emerging type of systems. There will be many systems that collectively gather information from user devices, e.g., smartphones, as well as from the IoT devices, e.g., sensor motes.

Although there have been many new projects in the area of IoT, including the design tool for IoT-integrated systems I proposed in Chapter 6, we are still in an early stage of the IoT technology. Only a few companies have achieved a business success based on IoT. An example is Nest, a company that builds smart sensors and various other products for home automation.[1] For example, Nest builds thermostats, embedded devices that are attached to walls at home to monitor various environmental variables including room temperature, humidity, gas levels and illumination. This collected information is then processed in the cloud to analyze the users behavioral patterns - when they go to work, when they come back, what is the room temperature they like, how often they ventilate the room, and so on. The analyzed user patterns are an integral part of making a smart home with more convenient interfaces and more energy-efficient home management.

Amazon is a rising player in the IoT market. They have a variety of 'thing' product lineups including the customer front-end devices, such as Kindle TV, Amazon Echo, and Amazon Dash. Users can speak to the device or just press a button in their living room to place orders, play music,

[1] They were recently acquired by Google, consequently changing their name to Google's Nest Labs.

or search interesting information. AWS can be used as the IoT backbone service. Their computing frameworks including Kinesis, Lambda, DynamoDB, ML, EMR and Redshift on the elastic EC^2 cloud computing infrastructure can serve the needs for data ingestion, data processing, data storage, machine learning, and analytics.

In academia, a number of research projects related to IoT have started recently. Filho et al. propose an integrated distributed system consisting of an Internet of Things (IoT) and a cloud computing infrastructure [100]. This system is used to understand the natural environment interdependencies (e.g., animals might affect water quality) to manage the environment through interventions (e.g., a catchment). It also supports high-level system specifications in the environmental science context to represent environmental science concepts. Ma et al. present a distribute database framework for IoT systems [214], with an efficient update and query index based on HBase, a key-value storage. This can support high insert throughput and provide efficient multi-dimensional simultaneous queries. There is a way to compose a new IoT service from existing services, called mashup [152]: it allows things to expose web service functionalities based on the legacy web mashup technology. The researchers also propose a cloud-based IoT mashup service model called IoT Mashup as a Service (IoTMaaS). This method can handle the heterogeneity of devices based on the model driven architecture principles and the cloud computing's computational scalability. In a recent research project, Hassan et al. develop a cloud-assisted IoT framework for healthcare in the smart city environment [132]. This framework aims to address the challenges in integrating IoTs and cloud computing in the healthcare domain, such as reliable transmission of vital sign data to cloud, dynamic resource allocation to facilitate seamless access and processing of IoT data, and effective data mining techniques.

The progress of IoT systems will impact various new areas. In the business area, IoT can help consumers making decisions through the augmented intelligence collected by the IoT sensors and devices. The Internet of Business Things (IoBT) will help companies achieve enhanced process optimization and efficiencies based on the data collected from the business environment. An increasing number of businesses will add sensors to people, places, processes and products for information collection and analysis, thus helping the businesses make better decisions.

Another trend involves the scale of IoT-based smart places that gets increasingly large. The vision of the smart home will be realized through new devices such as new generations of smart

home-appliances, entertainment devices, smartphones, and tablets. At the next level, smart buildings efficiently manage temperature, resources, and emergency cases using the information collected from the building. Nowadays, many companies and government departments are striving to make their cities smarter by more efficiently managing energy consumption, traffic, and healthcare. As the available power of the cloud computers keeps growing, the impact of IoT will also grow. In the end, a global-level of management for energy and resource will become available.

11.1.2 Altocumulus: An Intermediate Cloud System

The utilization of smart devices located at the edge of the cloud networks continues to accelerate, leading to new forms of heterogeneous cloud computing models. This trend has been influenced by two factors. First, the number of intelligent embedded devices in the cloud, such as smartphones and set-top boxes, has significantly increased. Second, cloud application executions like data processing and analytics are becoming more delegated from the clouds to the edge, thus making the transactions and communications in the cloud more local.

Together with Marcin Szczodrak, Richard Neill, and Luca Carloni, I envision that a new form of computing systems will emerge as one of the future MCC systems [165]. We call it Altocumulus, a new class of heterogeneous cloud computing which occurs at the edge of the cloud where it harvests the resources of edge devices, typically embedded systems, to enhance application capabilities and performance. By reporting on a number of prior studies, we claim that Altocumulus is well positioned to support all main cloud-computing service models: IaaS, PaaS, and SaaS. We present our vision on how Altocumulus could develop in the context of other cloud computing technologies by discussing different models of computation delegation. We conclude by presenting possible use-case scenarios and open areas of research.

11.1.2.1 Evolution of MCC

Due to the explosive proliferation of embedded devices, particularly mobile devices, three major factors have been introduced into the world of cloud systems. First, the number of nodes that send and receive data packets in the network has increased significantly [342; 246]. In particular, the number of smartphones in use has passed 1 billion in 2012 and is expected to reach 2 billion in 2015 [24]. This factor caused the change of the shape of data transmission in the

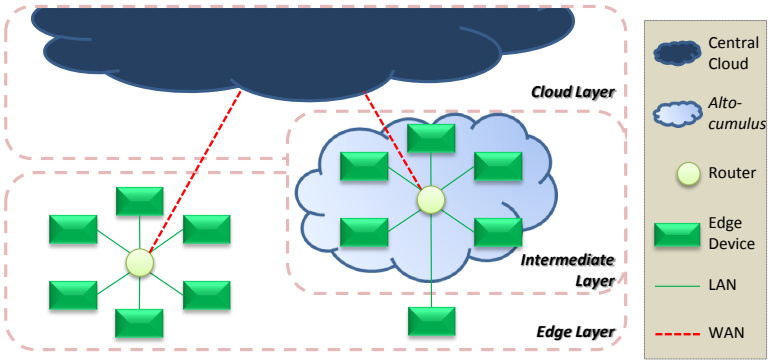
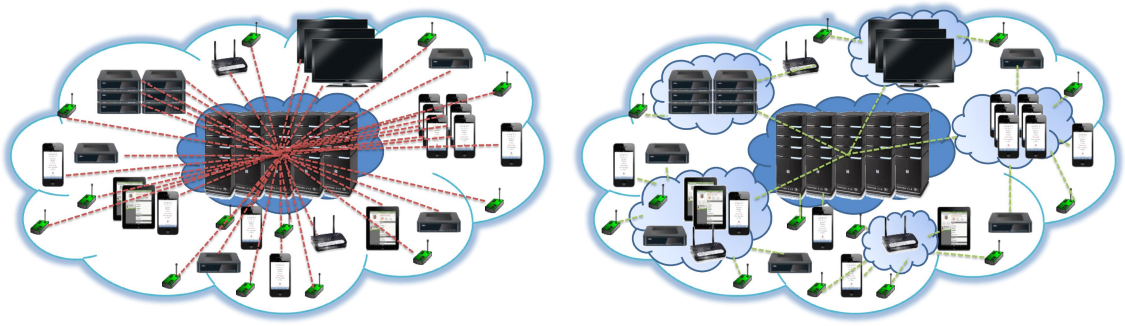


Figure 11.1: A cloud network topology with the intermediate layer introduced by Altocumulus.

cloud network from transactional computations to one with more interactive communications [78]. Second, the processing power of embedded devices continues to increase. In 2013, many set-top boxes (STBs) [187], smartphones [189], and TVs [211] have processors running with frequencies above 1GHz along with larger than 1GB main memory, equipped with hardware accelerators [336; 67]. Finally, the number of computation requests from each mobile devices stays often constant, or less intensive at times, while requiring instead shorter response times [154; 330; 238; 259].

Keeping pace with this trend, the cloud continues to adapt and evolve: more devices on the 'edge' of the cloud now spontaneously contribute to the computation the cloud solely used to process. As illustrated in Figure 11.1, the edge of the cloud is where the cloud network is connected to end users. We call *edge devices* the devices that are located in this area. Without physically relocating edge devices, their processing powers and networks can be used to form small clouds. From the network topology point of view, this newly formed small cloud is viewed as 'intermediate' to an edge device which accesses it, being located on the path from the cloud layer to that user device on the edge layer. In this network, computations required for cloud application execution can be partially delegated to these intermediate clouds, which consist primarily of edge devices. We argue that the formation of this intermediate cloud layer is emerging across a variety of systems. The edge devices located within intermediate clouds include mobile devices, mostly being smartphones and tablets, home appliances like smart TVs and STBs, and all other types of embedded devices such as routers and sensor motes.

We call the new type of cloud computing that has this phenomenon *Altocumulus*. Altocumulus



(a) Conventional cloud computing (b) Cloud computing with Altocumuli

Figure 11.2: The comparison of transaction flows in cloud computing.

means a cloud at an intermediate position[2]. We use the term 'cloud' to indicate the large, remote, and public cloud with powerful server machines in the data center. In contrast, Altocumulus may be described as a small-sized, and local private cloud powered by a network consisting of edge devices that are part of a large distributed embedded system. Due to the benefits of the embedded devices, Altocumulus in general has the following advantages over cloud computing.

1. **Network Traffic.** In cloud computing, shown in Figure 11.2(a), all the processing requests from the edge devices are converged to the cloud. Instead in Altocumulus, as illustrated in Figure 11.2(b) the edge devices send requests to their local clouds, or Altocumulus, and, if necessary, Altocumuli transfer these requests to the cloud in pre-processed or filtered forms. As a results, the total number of transactions (represented by the lines in the figure) is significantly reduced because Altocumulus serves as a local cloud using the resources from the embedded devices that exist physically close to the edges. This also results in shorter network-response time to the client devices.

2. **Cloud Costs.** Altocumulus is a local cloud voluntarily formed across the edge devices, which can partially, or often entirely, fill in the needs for the cloud services, including storage and computation services. Since Altocumulus is a newly emerging type of cloud computing and has not been used commercially, it has no cost model developed so far. Considering that currently available cloud services provided by cloud vendors usually adopt pay-as-you-go

[2]The definition of Altocumulus in the dictionary is "a globular cloud at an intermediate height of about 2400 to 6000 metres (8000 to 20,000 feet)" [130].

payment models, Altocumulus may take a similar cost model in the future [191]. Altocumulus can be, however, a more affordable solution that complements the cloud in the future because it leverages existing resources and the users also share their own resources.

3. **Resource Utilization.** To support a new generation of cloud applications, the cloud may require additional hardware system upgrades. In Altocumulus, however, the edge devices are periodically upgraded by their users, resulting in an automatic evolution in the hardware aspect. The fact that Altocumulus uses the existing embedded devices to provision computational power and storage spaces leads to better resource utilization overall. For example, wireless sensor mote devices generate many sensor data measurements that must be processed before higher level operations may be performed to interpret the data. A mechanism can be implemented on Altocumulus for filtering and pre-processing data to minimize network capacity and storage requirements at the cloud, thus optimizing resources for other cloud applications.

4. **Energy Efficiency.** Embedded systems, particularly mobile devices, are designed to be more energy efficient for using their resources than the server machines in the cloud [313; 245; 217]. Thus, it is likely that using embedded systems brings less energy consumption, as confirmed by a number of prior studies [244; 175; 241]. This may lead to interesting new research to address the following problems; (1) how efficient a cluster of embedded systems like Altocumulus is? and (2) how to optimally balance the loads across the conventional cloud and Altocumulus? [345]

In the following sections, we examine how Altocumulus fits in with various evolving technologies closely related to cloud computing. In Section 11.1.2.3, we shed a light on Altocumulus as the base system to support the three main cloud service models. Then, multiple combinations of computational delegation are presented in Section 11.1.2.4. Finally, in Section 11.1.2.5 we outline our vision for the future of Altocumulus.

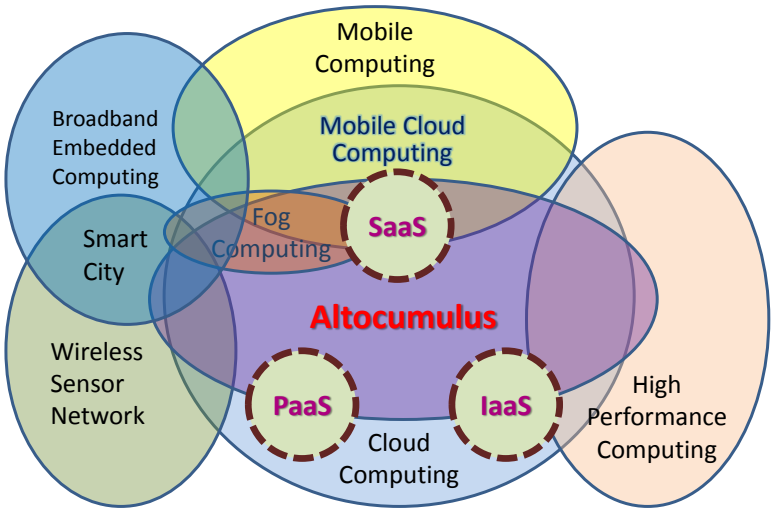


Figure 11.3: Taxonomy of cloud computing technologies.

11.1.2.2 Cloud Computing Classification

There have been diverse technologies related to cloud computing. A few important types are shown in Figure 11.3. This Euler diagram[3] describes the most significant intersections across the technology classes. Thus, it may not include a few newly emerging combinations among these classes, e.g., HPC applications on SaaS [71].

Among these technology areas, what gains ample attention is the combination of cloud computing and mobile computing, called Mobile Cloud Computing (MCC). In MCC, mobile devices usually send requests to the cloud to process certain computational problems or store the users' information. The user devices in MCC delegate heavy computations to the cloud due to mobile devices' limited resources in terms of computational power and battery capacity.

This heavy computation requirement of the cloud servers relates cloud computing to High Performance Computing (HPC). Certain cloud service vendors prepare server machines that are strong enough to support HPC applications and provide HPC cloud services, allowing their users to leverage the benefits of cloud computing: easy management, low cost, and elasticity [15].

Broadband Embedded Computing (BEC) is an emerging area of research to develop a computing platform that leverages both a collection of embedded devices and a broadband network

[3]Note that Figure 11.3 has no intersections among certain components and thus it is not a Venn diagram. A Venn diagram must present 2^n zones for n components while an Euler diagram may not.

connecting them. Our prior work in this area led us to envision the evolution of Altocumulus to support all main service models of cloud computing: IaaS, PaaS and SaaS. Specifically, as discussed next, we developed a processor virtualization technique, two distinct computational frameworks, a distributed file systems and various distributed applications.

Wireless Sensor Network (WSN) are combined to cloud computing, for instance by processing sensor data with Hadoop and HBase [351; 350]. The processing power of cloud computing opens new possibility for WSN applications.

The intersection of BEC and WSN is the area in which the base technologies for realizing the concept of Smart Cities are arising [268; 270]. The broadband network provides the accessibility to each household and the sensors collect the data necessary for building a smart city. On top of this groundwork, the processing power offers in-time analysis of the collected data by executing data mining and machine learning applications.

Finally, as shown in Figure 11.3 Altocumulus overlaps all the discussed computing technologies. Altocumulus also supports the full coverage of the three main service models of cloud computing, IaaS, PaaS, and SaaS, as discussed in the next section.

11.1.2.3 Altocumulus Service Models

Although there have been a multitude of efforts to extend cloud computing service models, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) are still recognized as the most fundamental ones.[4] Figure 11.4 shows the system stacks from both hardware and software perspectives and exemplary components of Altocumulus with regard to the three cloud service models. In this section, we delve into how Altocumulus supports each of these service models, based on our own experimental efforts on all of the models along with other closely related projects.

IaaS: Virtualization. IaaS is the most basic cloud service model where users buy virtual machine instances from the service providers instead of purchasing physical machines, which require installations, operations, and management. Virtualization and virtual machine management are the core features which enable the IaaS model and, as shown in Figure 11.4, virtualization is the layer

[4]In 2012, Network-as-a-Service (NaaS) and Communication-as-a-Service (CaaS) were officially added by International Telecommunication Union (ITU). There has been no platform running on the edge devices for these new service models, since the platforms that belong to Altocumulus so far have been focused more on computational models.

directly exposed to IaaS users.

A variety of efforts, including our experiments, have been made to provide virtualized resources from edge devices to the users. Our work on virtualizing edge devices' processors and providing them to a server machine is a heterogeneous approach to give an easy access to the edge devices' processing power [233]. Some view this as a distinct type of service model, such as Hardware as a Service (HaaS) since it offers an abstract interface to access hardware [294]. There is also a custom type of Virtual Machine Manager (VMM) that runs on the ARM processors [155]. This shows the possibility of the virtualization on the most popular embedded processor architectures, such as ARM.

Some foundations for the IaaS model in Altocumulus are provided by recent studies on the virtualization of the edge devices with ARM processors based on the two main hypervisors, KVM and Xen [81; 147]. Also, one of the industry's leading companies, VMWare, has created a platform to leverage the opportunities on mobile devices [45]. Meanwhile, processors for mobile computers, i.e., laptops, like Intel Atom processors are often used also for embedded systems and studies focusing on the virtualization of these mobile processors are expanding the virtualization aspect of IaaS [298; 297].

Based on virtualization techniques developed on edge devices, the on-going and future efforts for developing custom-design IaaS platforms and porting existing ones such as OpenStack, OpenNebula, or Eucalyptus [22; 21; 237] will pave the way to Altocumulus' IaaS service. Although virtualization and IaaS frameworks for edge devices are at their early stage, there is high potential for Altocumulus to serve as a new IaaS infrastructure, as the technology matures and new business models are developed.

PaaS: Computing and Storage Frameworks. PaaS offers a basic software stack and services to build customized applications based on the PaaS APIs. A PaaS service can be implemented based on three components: a web service engine, a data storage solution, and a computing framework [249]. Among these, we first concentrate on the computing framework for two reasons. First, in many platforms it supports the PaaS model as the most integral part of the backend implementation. Second, the computing framework itself can be viewed as a PaaS service, providing computing services through their own APIs.

There are a few custom computing frameworks available [37]. However, PaaS is to provide

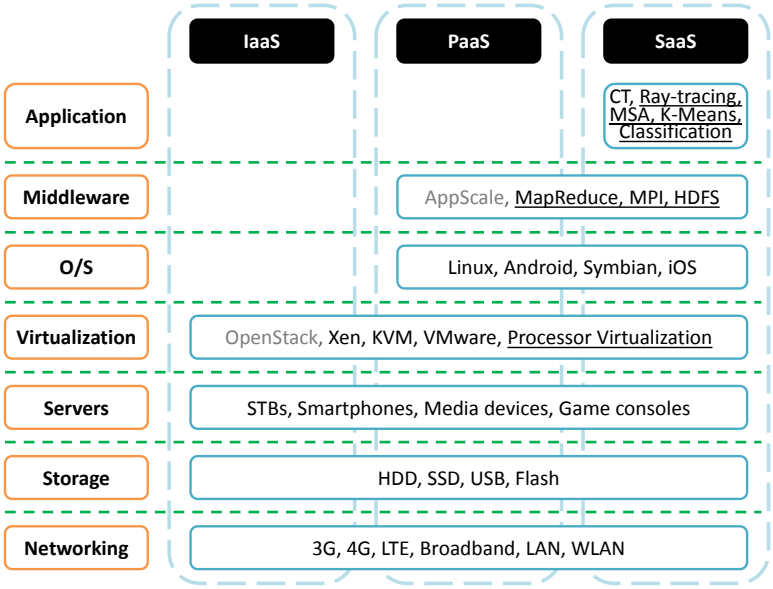


Figure 11.4: System stacks and example components for Altocumulus' service models. Work-in-progress and planned projects are written in gray and our own prior projects are underlined.

APIs for applications. It is important to support widely used APIs to gain wide acceptance from various PaaS users. This fact led us to focus on providing Altocumulus with the two most popular computing frameworks: MapReduce and Message Passing Interface (MPI).

MapReduce is one of the popular PaaS services which many scientists, researchers, and analysts use for big data analysis. Many cloud providers offer MapReduce as one of their core PaaS services [1]. We have ported Hadoop, an open source implementation of the MapReduce framework and a distributed file system, to a cluster of STBs, as discussed in Chapter 7. This work demonstrates the possibility that Altocumulus serves PaaS through the MapReduce framework and Hadoop Distributed File System (HDFS).[5] Another study implemented a custom MapReduce framework on a cluster of smartphones [171]. It is easy to predict that in the near future, we will see more experimental platforms that heterogeneously combine these edge devices.

MPI is a standardized message passing interface designed to be efficient, simple, and portable. While MPI has been a major framework for the HPC systems, there are also certain needs to have the same framework supported by cloud services as a PaaS model. For instance, a cloud service provider may want to offer a service over a variety of programming models [291]. Altocumulus

[5]Here, we use the three main cloud service models but some distinguish Storage-as-a-Service (STaaS) from PaaS.

can address these needs by leveraging recent MPI porting efforts to a variety of embedded devices: AppleTVs [109], PlayStation [300] and STBs [234].

Aside from computing frameworks, Altocumulus can also provide storage services. Our Hadoop porting to the STB cluster also provides HDFS as presented in Chapter 7. Continuous research on executing HBase over the edge devices is promising due to its linear scalability, consistency, and automatic failover, which are common requirements of many cloud-based service applications [4].

These frameworks are the deliverables to PaaS users in the Altocumulus service models as shown in Figure 11.4. Thanks to these efforts in harvesting the computational power of the edge devices, Altocumulus is well positioned to provide a new PaaS infrastructure.

SaaS: On-Demand Applications. In recent years many different applications have been ported to networks of embedded devices following an approach like Altocumulus. Figure 11.4 shows some of the applications we describe in this section.

Multiple Sequence Alignment (MSA) is a fundamental problem in bioinformatics that requires to arrange three or more sequences of DNA, RNA, or protein to identify regions of similarity [55]. We ported a MPI-based MSA application to a heterogeneous cluster of STBs and Blades [234]. We also ported and analyzed a Ray-Tracing application which runs a computation-intensive 3D graphics algorithm on the same cluster [233]. The MSA and Ray-Tracing examples follow the Altocumulus model, providing enhanced computational services based on the resource utilization of embedded devices at the edge network in conjunction with centralized server-cluster computation.

We also provided MapReduce-based applications that solve two key data mining algorithms, K-Means clustering and Classification [166]. These applications scale well across the Hadoop environment over a BEC system.

Certain research has focused on executing applications on a game console. A cone-beam CT image reconstruction application [184] and a Neuromorphic application [300] are developed to run on the Sony's PlayStation 3.

The number of SaaS applications has significantly increased based on the IaaS and PaaS services with the emergence of cloud computing. Similarly SaaS is also likely to thrive in Altocumulus once the development of IaaS and PaaS has made sufficient progress in this environment. In fact, the growth of SaaS for Altocumulus can be even more accelerated by taking advantage of recently proposed methodologies for the development of these applications [210; 301; 315].

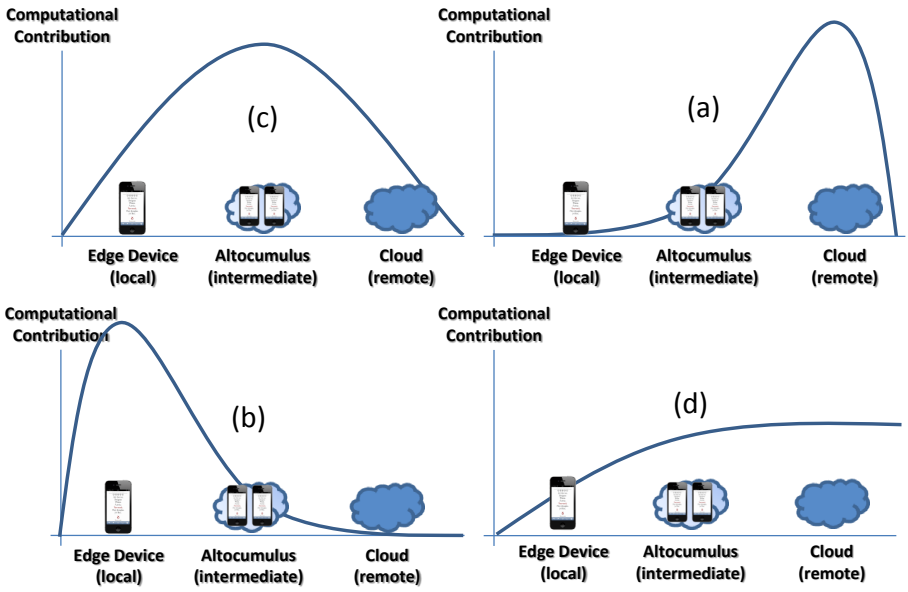


Figure 11.5: Computational delegation graphs.

11.1.2.4 Delegation of Computation

As Altocumulus takes part in cloud computing, particularly in the middle of the paths between the cloud and the edge devices, it is designed to serve both parts in-between. In this section, we discuss how Altocumulus can contribute to the distribution of the computation in the network.

Figure 11.5 shows four possible types of *computation delegation* models on an Altocumulus network. In each graph, the X axis denotes the topological distance from the edge device and the Y axis indicates the amount of computation processed by the nodes at the X distance for cloud or Altocumulus services.

Figure 11.5(a) illustrates the case when most of the computation is handled only by the cloud. This is the case of many currently existing MCC systems, where an increasing number of requests from thin clients on the edge are processed entirely by the cloud relying on the elasticity of its service provider.

In Figure 11.5(b) the edge devices resolve the computational needs without requesting for delegation. As the computational power of an edge device keeps growing, it becomes possible to process certain amount of computations in the device. An example is graphic-intensive offline-based game for mobile devices.

Figure 11.5(c) shows that the major amount of computation that happens in the network is

processed by Altocumulus, i.e., by the nodes in the middle of the path from the edge device to the cloud. In this configuration, Altocumulus serves as a local cloud which may reduce the cloud costs, network latency, and energy consumption [117].

Figure 11.5(d) indicates that the amounts of computation delegated to each Altocumulus and the cloud are approximately comparable. The uniform distribution of computation tasks across Altocumulus and the cloud has advantages for some applications, e.g., when the cloud is suffering from heavy workloads and balancing the loads between the cloud and Altocumulus draws the optimal performance in time, costs, or Quality of Service (QoS). The distribution of workloads, however, differs from the load balancing among the server nodes. Instead, Altocumulus could contribute to the cloud in a particular way as it filters, reduces, caches, or reproduces the information moving from the edge to the cloud. There is ample potential for this type of computational distribution and more specific examples on how Altocumulus can contribute are given in Section 11.1.2.5.

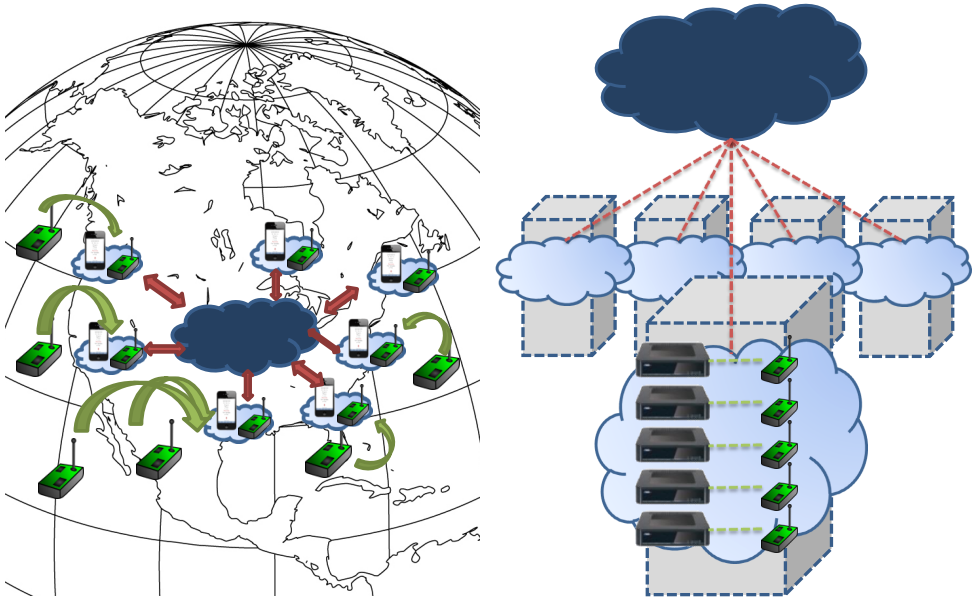
While Figure 11.5 provides a high-level abstraction of the main possible computation delegation models, there is a rich variety of possible combinations. Moreover, the suitability of each model is application specific and differs largely according to the configurations or status of the base infrastructures; i.e., the devices and networks. Therefore, no single form of delegation model can be vastly superior to all others or fit into diverse environments. To find the optimal proportion for computational delegation between the cloud and Altocumulus is one of the pivotal areas of future research.

11.1.2.5 Use-Case Scenarios of Altocumulus

The future success of the Altocumulus model is tied to the implementation of new mobile and wireless sensor network applications. For instance, we expect that a major focus activity will involve how to use emerging edge devices to enhance the processing performance of large-scale data analytics applications.

We discuss two use-case scenarios that fit to Altocumulus in terms of their layered, hierarchical designs. While each of these applications could be implemented also in other ways, designing them to run partially on Altocumulus facilitates the use of resources from the edge devices, thereby alleviating the computation and communication load for the cloud.

Each of these use-cases can be implemented also without Altocumulus, but designing to run



(a) Weather Data Analysis (b) Smart Buildings

Figure 11.6: Two Altocumulus use-case scenarios.

those applications on a cloud with Altocumulus will help the cloud to alleviate the overall stress from computation and communication, facilitating the use of resources from the edge devices.

Weather Data Analysis. This is an Altocumulus-suited version of the Weather Data Analysis example introduced in [340], which analyzes the National Climatic Data Center (NCDC) weather dataset. This centralized application processes the weather data, including sky ceiling heights, visibility distances, temperature, and atmospheric pressure, gathered from across the globe. The centralized data processing burdens the cloud with a large amount of work that comes from tens of thousands of weather stations. With Altocumulus, this application can be implemented in a hierarchical fashion as illustrated in Figure 11.6(a). Each Altocumulus collects data from nearby sensors. Then, only the summarized information necessary to analyze country-wide or worldwide weather statistics is transferred from Altocumulus to the cloud. For example, the *MaxTemperature* command that finds the highest temperature can be processed easily with Altocumulus. Each Altocumulus processes the weather data from local sensors and reports summarized information that includes the local maximum temperature to the cloud. Then, the cloud finds the highest temperature among the reports from Altocumuli. This hierarchical approach offers lesser burdens on the cloud

and shorter processing time due to its parallelized analysis of the data.

Smart Buildings. The goal of this application is twofold. First, each building manages the use of its resources such as electricity, water, and gas, in a smart way. Second, a central control system monitors the resources in each building for the city-level management, e.g., forecasting power outages, improving resource distribution planning, and saving the energy. We address this problem by mapping Altocumulus to a building and the cloud to the city. Figure 11.6(b) shows Altocumulus configured for each building having a STB and sensors placed in every house. The information gathered from the sensors is stored in Altocumulus and analyzed for understanding the users' behavior patterns by executing machine learning algorithms. Using the analyzed user behavior patterns, Altocumulus manages the light and temperature control systems to reduce energy consumption of the houses in the building. For energy-consumption tracking and planning at the city-level, the Altocumulus of each building reports to the cloud statistical information about consumed resources and observed patterns. The collected building data is processed to extract the information necessary to the city management.

11.1.2.6 Related Work

A study on the clustering devices on Field Area Network called Fog Computing [57] embraces the ideas of heterogeneity and mobility. Carmen [182] is a system that manages the mobile connectivity, focusing on the transitions between networks and devices, to improve the mobile user experience. These studies, however, do not focus on leveraging the computational power of these devices. Instead, Altocumulus is based precisely on the idea that edge devices actively engage in the heavy computations of the cloud, taking parts, or often the whole, of the computational tasks.

11.1.2.7 Concluding Remarks

The evolution of cloud systems is toward greater heterogeneity and distribution of computation between a cloud data-center and embedded devices at the network edge. The computational distribution may be delegated dynamically based on the application domain to optimize computation and resource utilization. To support this ongoing evolution of cloud computing and applications, we proposed Altocumulus as an intermediate cloud computing infrastructure that simplifies this dynamic distribution and delegation of computation. We illustrated the Altocumulus concept by presenting a

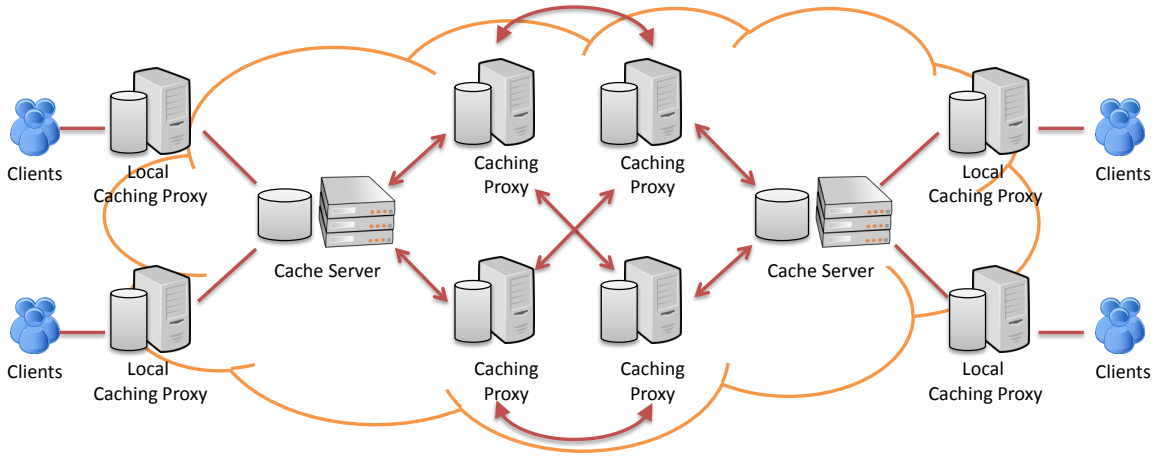


Figure 11.7: A content delivery network based on caching proxy mesh.

number of experimental systems that have been developed recently, including systems based on the MapReduce and MPI programming models. Finally, we have sketched some ongoing and planned Altocumulus projects.

11.2 Future Design Tools for MCC

The development of emerging generations of MCC systems will need adequate design tools. So far, the design tools have evolved according to the MCC systems' transformation. Future design tools for MCC systems will continue to expand their features in consonance with the characteristics of the future MCC systems. As the future generation of MCC systems are expected to grow toward even larger scalability and greater heterogeneity, the design tools to simulate these systems will have to support such a unprecedented level of scalability and heterogeneity. Additionally, some emerging systems have a different system architecture or new types of networks that the current design tools cannot support. Figure 11.7 illustrates the architecture of an example Content Delivery Network (CDN). A CDN could be used as part of new MCC systems that will have to be supported by future design tools. Likewise, supporting a new system architecture, e.g., high-speed instance launching for Altocumulus where a large number of embedded devices can dynamically join and leave the network.

Among the expected requirements of new design tools for future MCC systems, there are:

- **Larger Scalability.** Some future systems will comprise hundreds of millions of computing nodes, e.g., IoT devices and sensors. One way to support this level of scalability will be simulating some groups of nodes at a more abstracted level. For instance, after learning the scaling behavior of outgoing network packets from a particular device group, scaling up could be represented by generating emulated packets as if they come from the added devices.

- **Higher Heterogeneity.** Future MCC systems have a variety of different computing nodes including smart devices, home alliances, smart vehicles, outdoor stationary devices, wearable equipments, sensors, and home controlling systems. These computing nodes have distinct processing cores, different network characteristics (e.g., bandwidths, latencies, and error rates), and different peripherals configured in each node.

- **New Architectures.** Future MCC systems can comprise new network and system architectures, e.g., Altocumulus, CDN, or Software-Defined Network (SDN), as one of the components for the entire system. The future design tools for such systems will need to support these new architectures, dealing with the details of each element. For instance, a future design tool can incorporate a SDN simulator.

- **Hardware-in-the-loop Simulation.** In addition to the complex design of the future MCC systems, the design tools have to support the smooth, incremental migrations of the developed software from the simulation environment to the physical hardware devices. Hardware-in-the-loop (HIL) simulation is a technique to develop and test systems by emulating certain portions of the system, e.g., input modules. HIL simulation could be applied to the MCC design tools, e.g., by replacing a node with a physical hardware device inside a simulated MCC environment. For those future MCC systems that have large-scale, heterogeneous components and a distributed network, HIL simulation can enhance the quality of testing, speed up the debugging process, and ease the migration from simulation to deployment on physical hardware.

- **Analytical Features.** Many virtual platform-based design tools offer some analytical functions which calculate the values of some useful information, e.g., execution times, power consumption, or heat dissipation, expected in a single hardware device. This information from one device can also be used to estimate the information associated to the entire distributed

system. For instance, the sum of each device's power consumption corresponds to the overall power consumed by the whole system. Moreover, future design tools can obtain more sophisticated information such as the installation and deployment costs or system availability estimations. For instance, a tool can calculate the cost for buying, installing, and maintaining the computing and network nodes in the system as the scale of the system changes. Likewise, a tool can anticipate the availability of the system by utilizing the probability-based reliability of each computing and network node and the replication plan of the system.

Chapter 12

Conclusions

Thanks to the success of the cloud computing and mobile computing technologies, MCC is becoming increasingly popular. To keep its success in the future computation world which requires an ever-increasing system scale and exceeding heterogeneity, we need tools well suited to help the design, development, and test of the newly designed MCC systems. In this thesis, I have presented NETSHIP, and its subsequent improvements to support the simulation of mobile GPUs and IoT systems. NETSHIP is a scalable networked virtual platform that effectively supports the specification, design, and testing of MCC systems with many heterogeneous components.

Using NETSHIP, I have developed three unique MCC application systems. First, I have presented a broadband embedded computing system. It is a distributed computing system that runs the Hadoop computing framework. It is heterogeneous because it combines a cluster of blade servers and a cluster of embedded devices. Next, I developed the LN-Annote system that enables the personal information service on the mobile devices by extracting and keeping personal information locally. Finally, I developed an audio stream retrieval system where the users find the information on the original media content by sending a segment of the input audio stream they have. Each of these application systems is based on the general MCC architecture, where the client devices access the cloud server to use its higher computational power, larger storage capacity, and more affordable energy budget. I used NETSHIP to develop the MCC client applications and perform design-space exploration and optimization.

As a result, I have invented three distinct types of optimization techniques for the distribution of computations in MCC systems. First, reverse distributed offloading is where the client devices

take the bulk of the computation. Second, algorithm-division offloading is a method to divide the core algorithm into two halves so that one half can run on the cloud while the other half can run on each device. Third, query-division offloading sends only certain queries to the cloud servers while processing the rest locally. This spectrum of optimization techniques for distinct distribution of computations can be applied also to many other types of systems, thus increasing execution performance, energy efficiency, and resource utilization.

Throughout the development of the design tools and the applications, I have presented the iterative co-development process. By showing how each project contributes to another, I gave a guidance for the users who develop both design tools and applications. In particular, this co-development process can help the users organize the requirements for the new design tools based on their feedbacks from developing applications with previous versions of the tool. On the other hand, I have shown that development of new applications can be expedited by using the design tools and how the tools can play a key role in developing and optimizing MCC applications.

In general, I believe that this line of studies on MCC has opened a new avenue of research. The insights from designing and developing new MCC application systems can help the development of large-scale, heterogeneous systems in the future. Particularly, the invention of the optimization techniques not only can help new systems perform in a more optimized way but it can also become a new research area in itself. I expect the development of many new optimization techniques of this sort to continue in the future.

Part IV

Bibliography

Bibliography

[1] Amazon Elastic MapReduce (aws.amazon.com/elasticmapreduce).

[2] Amazon Product Ads (www.amazon.com/productads).

[3] Apache hadoop (hadoop.apache.org).

[4] Apache HBase (hbase.apache.org).

[5] Apple Siri (www.apple.com/ios/siri).

[6] AWS Total Cost of Ownership Calculator (awstcocalculator.com).

[7] Backport (backport-jsr166.sourceforge.net).

[8] Cablelabs (www.cablelabs.com).

[9] CUDA (developer.nvidia.com/cuda).

[10] Facebook Ads (www.facebook.com/ads).

[11] Google Adsense (www.google.com/adsense).

[12] Google Android (developer.android.com).

[13] Google Now (www.google.com/landing/now).

[14] Google Play Sound Search: (goo.gl/ahpvyO).

[15] HPC on AWS (aws.amazon.com/hpc-applications).

[16] i-YunoMG CaptionCast (www.captioncast.com).

[17] NYC Vision Zero (`www.nyc.gov/visionzero`).

[18] Open MPI (`www.open-mpi.org`).

[19] Open Virtual Platforms (`www.ovpworld.org`).

[20] OpenCV (`opencv.org`).

[21] OpenNebula (`www.opennebular.org`).

[22] OpenStack (`www.openstack.org`).

[23] Retrotranslator (`retrotranslator.sourceforge.net`).

[24] Strategy Analytics (`strategyanalytics.com`).

[25] The Flickr API (`www.flickr.com/services/api`).

[26] The Picasa Web Albums Data API (`developers.google.com/picasa-web`).

[27] Tru2way (`www.tru2way.com`).

[28] D. Aarno and J. Engblom. *Software and System Development using Virtual Platforms*. Morgan Kaufmann Publishers, 2014.

[29] D. Aarno and J. Engblom. *Software and System Development using Virtual Platforms: Full-System Simulation with Wind River Simics*. Elsevier, Sept. 2014.

[30] S. Abolfazli et al. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *IEEE Comm. Surveys Tutorials*, 16(1):337–368, Feb. 2014.

[31] S. Abolfazli et al. Rich mobile applications: Genesis, taxonomy, and open issues. *Journal of Network and Computer Applications*, 40(0):345 – 362, Apr. 2014.

[32] P. Abrahamsson et al. Effort prediction in iterative software development processes – incremental versus global prediction models. In *Proc. of the Int. Symp. on Empirical Software Engineering and Measurement*, pages 344–353, Sept. 2007.

[33] E. Alba. Intelligent systems for smart cities. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 707–722, July 2015.

[34] D. Albesano et al. Adaptation of artificial neural networks avoiding catastrophic forgetting. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 1554–1561, July 2006.

[35] R. Amorim et al. Comparing CUDA and OpenGL implementations for a Jacobi iteration. In *Proc. of the Int. Conf. on High Perf. Comp. and Sim.*, June 2009.

[36] J. Arndt and C. Haenel. *Pi-Unleashed*. 2001.

[37] M. Y. Arslan et al. Computing while charging: building a distributed computing infrastructure using smartphones. In *Proc. of the Int. Conf. on Emerging Net. Experiments and Tech.*, pages 193–204, Dec. 2012.

[38] H. Assal et al. Partnering enhanced-NLP with semantic analysis in support of information extraction. In *Proc. of the Int. Workshop on Ontology-Driven SW Engineering*, pages 9:1–9:7, Oct. 2010.

[39] M. Aswad et al. Context aware accidents prediction and prevention system for VANET. In *Proc. of the Int. Conf. on Context-Aware Syst. and App.*, pages 162–168, Oct. 2014.

[40] C. Atkinson and O. Hummel. Iterative and incremental development of component-based software architectures. In *Proc. of the Symp. on Component Based Software Engineering*, pages 77–82, June 2012.

[41] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan. Advancing the state of mobile cloud computing. In *Proc. of the Workshop on Mobile Cloud Comp. and Services*, pages 21–28, June 2012.

[42] A. Bakshi et al. MILAN: A model based integrated simulation framework for design of embedded systems. In *Proc. of the Workshop on Optimization of Middleware and Dist. Syst.*, pages 82–93, June 2001.

[43] S. Bangay. Experiences in porting a virtual reality system to Java. In *Proc. of the Int. Conf. on Comp. Graphics, Virtual Reality and Visualisation*, pages 33–37, Nov. 2001.

[44] W. C. Barker and E. B. Barker. SP 800-67 Rev. 1. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. Technical report, Jan. 2012.

[45] K. Barr et al. The VMware mobile virtualization platform: is that a hypervisor in your pocket? *SIGOPS Oper. Syst. Rev.*, 44(4):124–135, Dec. 2010.

[46] A. Bartolini et al. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proc. of the Symp. on Great Lakes Symposium on VLSI*, pages 311–316, May 2010.

[47] M. Bartsch and G. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *Trans. on Multimedia*, 7(1):96–104, Feb. 2005.

[48] H. Bauer et al. The supercomputer in your pocket. *McKinsey on Semiconductors*, pages 1–1, Sept. 2012.

[49] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 12 1966.

[50] R. G. Beausoleil. Large-scale integrated photonics for high-performance interconnects. *Journalof Emerging Tech. Comp. Syst.*, 7(2):6:1–6:54, July 2011.

[51] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proc. of the USENIX Annual Technical Conf.*, pages 41–46, Feb. 2005.

[52] L. Benenson et al. Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. *Data Center Efficiency Assessment*, pages 1–34, Aug. 2014.

[53] N. Bertin and A. D. Cheveigné. Scalable metadata and quick retrieval of audio signals. In *Proc. of the Int. Conf. on Music Info. Retrieval*, pages 238–244, Sept. 2005.

[54] R. Bifulco et al. Scalability of a mobile cloud management system. In *Proc. of the Workshop on Mobile Cloud Comp.*, pages 17–22, Aug. 2012.

[55] Y. Bilu, P. K. Agarwal, and R. Kolodny. Faster algorithms for optimal multiple sequence alignment based on pairwise comparisons. *Trans. on Comp. Bio. and Bioinfo.*, 3(4):408–422, Oct. 2006.

[56] B. Boehm. A view of 20th and 21st century software engineering. In *Proc. of the Int. Conf. on Software Engineering*, pages 12–29, May 2006.

[57] F. Bonomi et al. Fog computing and its role in the internet of things. In *Proc. of the workshop on Mobile cloud comp.*, pages 13–16, Aug. 2012.

[58] R. Cai, C. Zhang, L. Zhang, and W.-Y. Ma. Scalable music recommendation by search. In *Proc. of the Int. Conf. on Multimedia*, pages 1065–1074, Sept. 2007.

[59] G. Cameron et al. Paramics: Moving vehicles on the connection machine. In *Proc. of the Conf. on Supercomp.*, pages 291–300, Nov. 1994.

[60] P. Cano et al. A review of algorithms for audio fingerprinting. In *Proc. of the Int. Workshop on Multimedia Signal Proc.*, pages 169–173, Dec. 2002.

[61] J. Ceng et al. A high-level virtual platform for early MPSoC software development. In *Proc. of the Int. Conf. on Hardware/Software Codesign and Syst. Synthesis*, pages 11–20, 2009.

[62] T.-K. Chang. A secure cloud-based payment model for m-commerce. In *Proc. of the Int. Conf. on Parallel Proc.*, pages 1082–1086, Oct. 2013.

[63] M. A. Chauhan. A reference architecture for providing tools as a service to support global software development. In *Proc. of the WICSA Companion Volume*, pages 16:1–16:6, Apr. 2014.

[64] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Comm. of ACM*, 28(10):1030–1044, Oct. 1985.

[65] S. Chen, Y. Wang, and M. Pedram. Concurrent placement, capacity provisioning, and request flow control for a distributed cloud infrastructure. In *Proc. of the Conf. on Design, Automation and Test in Europe Conf.*, pages 279:1–279:6, Mar. 2014.

[66] S. Chen, Y. Wang, and M. Pedram. Optimal offloading control for a mobile device based on a realistic battery model and semi-Markov decision process. In *Proc. of the Int. Conf. on Computer-Aided Design*, pages 369–375, Nov. 2014.

[67] K.-T. Cheng and Y.-C. Wang. Using mobile GPU for general-purpose computing - a case study of face recognition on smartphones. In *Proc. of the Int. Symp. on VLSI Design, Automation and Test*, pages 1–4, Apr. 2011.

[68] L. Chiticariu et al. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proc. of the Conf. on Empirical Methods in Natural Language Proc.*, pages 1002–1012, Oct. 2010.

[69] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proc. of the Workshop on Mobile Cloud Comp. & Services: Social Networks and Beyond*, pages 7:1–7:5, June 2010.

[70] A. L. chun Wang. An industrial-strength audio search algorithm. In *Proc. of the Int. Conf. on Music Info. Retrieval*, Oct. 2003.

[71] P. Church et al. Toward exposing and accessing HPC applications in a SaaS cloud. In *Proc. of the Int. Conf. on Web Services*, pages 692–699, June 2012.

[72] A. Cidon et al. MARS: Adaptive remote execution for multi-threaded mobile devices. In *Proc. of the Workshop on Networking, Syst., and App. on Mobile Handhelds*, pages 1:1–1:6, Oct. 2011.

[73] S. Collange et al. Barra: A parallel functional simulator for GPGPU. In *Proc. of the Int. Symp. on Modelling, Analysis and Sim. of Comp. and Telecomm. Syst.*, pages 351–360, Aug. 2010.

[74] M. Collins. Discriminative training methods for Hidden Markov Models: Theory and experiments with Perceptron algorithms. In *Proc. of Conf. on Empirical Methods in NLP*, pages 1–8, July 2002.

[75] R. Collobert et al. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, Nov. 2011.

[76] R. Coop and I. Arel. Mitigation of catastrophic interference in neural networks using a fixed expansion layer. In *Proc. of the Int. Symp. on Circuits and Systems*, pages 726–729, Aug. 2012.

[77] R. Coop and I. Arel. Mitigation of catastrophic forgetting in recurrent neural networks using a Fixed Expansion Layer. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 1–7, Aug. 2013.

[78] I. D. Corporation. Extracting value from chaos, June 2011.

[79] L. Cosseboom. Asia is dominating the mCommerce market, puts us and europe to shame. *Tech in Asia*, pages 1–1, Sept. 2014.

[80] C. Dall and J. Nieh. KVM for ARM. In *Proc. of the Linux Symp.*, pages 45–56, July 2010.

[81] C. Dall and J. Nieh. KVM for ARM. In *Proc. of the Linux Symp.*, pages 45–56, July 2010.

[82] M. D'Angelo et al. A simulator based on QEMU and SystemC for robustness testing of a networked Linux-based fire detection and alarm system. In *Proc. of the Conf. on* $ERTS^2$, pages 1–9, Feb. 2012.

[83] L. David and I. Puaut. Static determination of probabilistic execution times. In *Proc. of the Euromicro Tech. Committee on Real-Time Syst.*, pages 223–230, June 2004.

[84] B. De Carolis, I. Mazzotta, N. Novielli, and S. Pizzutilo. Social robots and ECAs for accessing smart environments services. In *Proc. of the Int. Conf. on Advanced Visual Interfaces*, pages 275–278, May 2010.

[85] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. of the ACM*, 51(1):107–113, Jan. 2008.

[86] D. Diamantopoulos et al. Plug&chip: A framework for supporting rapid prototyping of 3D hybrid virtual SoCs. *Trans. on Embedded Comp. Syst.*, 13(5s):168:1–168:25, Dec. 2014.

[87] Y. Ding et al. Nao: A framework to enable efficient mobile offloading. In *Proc. of the Workshop on Posters and Demos Track*, pages 8:1–8:2, Dec. 2011.

[88] A. Dou et al. Misco: a MapReduce framework for mobile systems. In *Proc. of the 3rd Int. Conf. on Pervasive Tech. Related to Assistive Environments*, pages 32:1–32:8, June 2010.

[89] K. Duan et al. Multi-category classification by soft-max combination of binary classifiers. In *Proc. of the Int. Conf. on Multiple Classifier Systems*, pages 125–134, June 2003.

[90] P. R. Elespuru, S. Shakya, and S. Mishra. MapReduce system over heterogeneous mobile devices. In *Proc. of the Int. Workshop on SW Tech. for Embedded and Ubiquitous Syst.*, pages 168–179, Nov. 2009.

[91] J. Engblom. System simulation is a way to stress test your IoT application. *Embedded Computing Design*, pages 1–1, May 2015.

[92] X. Enhui, F. Yan, and W. Rui. Road safety analysis of long downgrade highway based on the property of vehicle brakes heat-resistant. In *Proc. of the Int. Conf. on Measuring Tech. and Mechatronics Automation*, volume 3, pages 258–261, Mar. 2010.

[93] D. Erhan et al. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, Mar. 2010.

[94] J. Eriksson et al. COOJA/MSPSim: Interoperability testing for wireless sensor networks. In *Proc. of the Int. Conf. on Sim. Tools and Tech.*, pages 27:1–27:7, Mar. 2009.

[95] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proc. of the Int. Symp. on Microarchitecture*, pages 449–460, Dec. 2012.

[96] J. Fang, A. Varbanescu, and H. Sips. A comprehensive performance comparison of cuda and opencl. In *Proc. of the Int. Conf. on Parallel Processing*, pages 216–225, Sept. 2011.

[97] N. Farooqui et al. A framework for dynamically instrumenting GPU compute applications within GPU Ocelot. In *Proc. of the Workshop on General Purpose Proc. using GPUs*, pages 9:1–9:9, Mar. 2011.

[98] V. Farrell, G. Farrell, K. Mouzakis, C. Pilgrim, and P. Byrt. PICTIOL: A case study in participatory design. In *Proc. of the Conf. on Computer-Human Interaction: Design: Activities, Artefacts and Env.*, pages 191–198, Nov. 2006.

[99] T. Fei, L. SunDong, and G. Sen. A novel method of crowd estimation in public locations. In *Int. Conf. on FBIE*, pages 339–342, Dec. 2009.

[100] R. V. R. Filho, B. Porter, and G. Blair. Environmental IoT: Programming cyber-physical clouds with high-level system specifications. In *Proc. of the Int. Conf. on Utility and Cloud Comp.*, pages 947–950, Dec. 2014.

[101] M. Fink, M. Covell, and S. Baluja. Mass personalization: social and interactive applications using sound-track identification. *Multimedia Tools and Applications*, 36(1-2):115–132, 2008.

[102] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proc. of the Annual Meeting on Assoc. for Comp. Linguistics*, pages 363–370, June 2005.

[103] J. Foote. An overview of audio information retrieval. *Multimedia Syst.*, 7(1):2–10, Jan. 1999.

[104] J. Forney, G.D. The Viterbi algorithm. *Proc. of the IEEE*, 61(3):268–278, Mar. 1973.

[105] M. Fresko, B. Rosenfeld, and R. Feldman. A hybrid approach to NER by MEMM and manual rules. In *Proc. of the Int. Conf. on Info. and Knowledge Management*, pages 361–362, Oct. 2005.

[106] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proc. of the Int. Conf. on Software Engineering*, pages 387–396, May 2004.

[107] S. Frølund and P. Garg. Design-time simulation of a large-scale, distributed object system. *Trans. on Modeling and Comp. Simulation*, 8(4):374–400, Oct. 1998.

[108] L. Fu, H.-H. Hsu, and J. C. Principe. Incremental backpropagation learning networks. *Trans. on Neural Network*, 7(3):757–761, May 1996.

[109] K. Fürlinger, C. Klausecker, and D. Kranzlmüller. The AppleTV-Cluster: Towards energy efficient parallel computing on consumer electronic devices. In *Proc. of the Int. Conf. on Inf. and Comm. on Tech. for the Fight against Global Warming*, pages 1–9, Aug. 2011.

[110] M. H. Gail et al. A solution to the generalized birthday problem with application to allozyme screening for cell culture contamination. *Journal of Applied Probability*, 16(2):242–251, June 1979.

[111] B. Gao et al. From mobiles to clouds: Developing energy-aware offloading strategies for workflows. In *Proc. of the Int. Conf. on Grid Comp.*, pages 139–146, Sept. 2012.

[112] Gartner. Forecast: Devices by operating system and user type, worldwide, Oct. 2013.

[113] Gartner. Forecast overview: Public cloud services, worldwide, Feb. 2013.

[114] F. Gatta et al. An embedded 65 nm CMOS baseband IQ 48 MHz-1 GHz dual tuner for DOCSIS 3.0. *Comm. Mag.*, 48(4):88–97, Apr. 2010.

[115] Y. Ge et al. A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In *Proc. of the Int. Symp. on Low Power Electronics and Design*, pages 279–284, July 2012.

[116] D. Geerts, R. Leenheer, D. De Grooff, J. Negenman, and S. Heijstraten. In front of and behind the second screen: Viewer and producer perspectives on a companion app. In *Proc. of the Int. Conf. on Interactive Experiences for TV and Online Video*, pages 95–102, June 2014.

[117] E. Gelenbe, R. Lent, and M. Douratsos. Choosing a local or remote cloud. In *Proc. of the Int. Symp. on Net. Cloud Comp. and App.*, pages 25–30, Aug. 2012.

[118] P. Gill et al. Youtube traffic characterization: A view from the edge. In *Proc. of the Internet Measurement Conf.*, pages 15–28, Oct. 2007.

[119] E. Giordano et al. MoViT: the mobile network virtualized testbed. In *Proc. of the Int. Workshop on Vehicular Inter-Net., Syst., and App.*, pages 3–12, June 2012.

[120] B. Girod et al. Mobile visual search. *IEEE Signal Proc. Magazine*, 28(4):61–76, July 2011.

[121] M. M. Gomes et al. Future directions for providing better IoT infrastructure. In *Proc. of the Int. Conf. on Pervasive and Ubiquitous Comp.*, pages 51–54, Sept. 2014.

[122] B. Goodrich and I. Arel. Unsupervised neuron selection for mitigating catastrophic forgetting in neural networks. In *Proc. of the Int. Symp. on Circuits and Systems*, pages 997–1000, Aug. 2014.

[123] N. Govindarajan et al. Event processing across edge and the cloud for internet of things applications. In *Proc. of the Int. Conf. on Management of Data*, pages 101–104, Dec. 2014.

[124] R. L. Graham, T. S. Woodall, and J. M. Squyres. Open MPI: a flexible high performance MPI. In *Proc. of the Int. Conf. on Parallel Proc. and Applied Math.*, pages 228–239, Sept. 2006.

[125] R. Gray. Vector quantization. *Acoustics, Speech, and Signal Proc. Magazine*, 1(2):4–29, Apr. 1984.

[126] A. Greenberg et al. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comp. Comm. Review*, 39(1):68–73, Dec. 2008.

[127] A. Greenberg et al. Towards a next generation data center architecture: Scalability and commoditization. In *Proc. of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 57–62, Aug. 2008.

[128] T.-M. Grønli, J. Hansen, and G. Ghinea. Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments. In *Proc. of the 3rd Int. Conf. on Pervasive Tech. Related to Assistive Env.*, pages 45:1–45:8, June 2010.

[129] J. Guo et al. Named entity recognition in query. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 267–274, July 2009.

[130] HarperCollins Publishers Ltd. *The Collins English Dictionary*.

[131] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, Mar. 1979.

[132] M. M. Hassan, H. S. Albakr, and H. Al-Dossari. A cloud-assisted internet of things framework for pervasive healthcare in smart city environment. In *Proc. of the Int. Workshop on Emerging Multimedia Applications and Services for Smart Cities*, pages 9–13, Nov. 2014.

[133] M. M. Hassan, H. S. Albakr, and H. Al-Dossari. A cloud-assisted IoT framework for pervasive healthcare in smart city environment. In *Proc. of Int. Workshop on Emerging Multimedia App. and Services for Smart Cities*, pages 9–13, Nov. 2014.

[134] B. He et al. Mars: a MapReduce framework on graphics processors. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 260–269, Oct. 2008.

[135] M. Heinrich and M. Gaedke. Data binding for standard-based web applications. In *Proc. of the Symp. on Applied Comp.*, pages 652–657, Mar. 2012.

[136] S. Hennig et al. User driven evolution of user interface models - the FLEPR approach. In *Proc. of the Int. Conf. on Human-Computer Interaction*, pages 610–627, Sept. 2011.

[137] J. Hensley, J. Isidoro, and A. J. Preetham. Combining computer vision and physics simulations using GPGPU. In *SIGGRAPH Sketches*, pages 1–1, Aug. 2007.

[138] C. Herley. Accurate repeat finding and object skipping using fingerprints. In *Proc. of the ACM Int. Conf. on Multimedia*, pages 656–665, Nov. 2005.

[139] J. C. Herrera et al. Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. *Transportation Research Part C: Emerging Tech.*, 18(4):568 – 583, 2010.

[140] K. Hillesland. OpenGL and DirectX. In *SIGGRAPH Asia Courses*, pages 10:1–10:79, Nov. 2013.

[141] S. Hong and H. Kim. An integrated GPU power and performance model. In *Proc. of the Int. Symp. on Comp. Arch.*, pages 280–289, June 2010.

[142] S. Howard and J. Martin. DOCSIS performance evaluation: piggybacking versus concatenation. In *Proc. of Southeast Regional Conf.*, volume 2, pages 43–48, Mar. 2005.

[143] Z.-M. Hsu, J.-C. Yeh, and I.-Y. Chuang. An accurate system architecture refinement methodology with mixed abstraction-level virtual platform. In *Proc. of Design, Automation and Test in Europe Conf.*, pages 568–573, Mar. 2010.

[144] Z.-M. Hsu, J.-C. Yeh, and I.-Y. Chuang. An accurate system architecture refinement methodology with mixed abstraction-level virtual platform. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 568–573, Mar. 2010.

[145] E. H. Huang et al. Improving word representations via global context and multiple word prototypes. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 873–882, July 2012.

[146] G. C. Hunt and M. L. Scott. The Coign automatic distributed partitioning system. In *Proc. of the Symp. on Operating Syst. Design and Impl.*, pages 187–200, Feb. 1999.

[147] J.-Y. Hwang et al. Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In *Consumer Comm. and Net. Conf.*, pages 257–261, Jan. 2008.

[148] IDC. Extracting value from chaos, June 2011.

[149] IDC. Smart connected device tracker, Mar. 2013.

[150] IDC. Worldwide datacenter census and construction 2014-2018 forecast: Aging enterprise datacenters and the accelerating service provider buildout, Oct. 2014.

[151] H. Ikeda et al. Abnormal incident detection system employing image processing technology. In *Proc. of the Int. Conf. on Intelligent Transportation Syst.*, pages 748–752, Oct. 1999.

[152] J. Im, S. Kim, and D. Kim. IoT mashup as a service: Cloud-based mashup service for the internet of things. In *Proc. of the Int. Conf. on Services Comp.*, pages 462–469, 2013.

[153] R. M. Ingle et al. An advanced traffic management system simulator for intelligent vehicle-highway systems research. In *Proc. of the Conf. on Winter Simulation*, pages 1455–1460, Dec. 1994.

[154] A. Inoue et al. Mobile internet-access behavior analysis. In *Proc. of the Int. Conf. on SW Eng., Artificial Intelligence, Net. and Parallel Dist. Comp.*, pages 766–770, Aug. 2012.

[155] H. Inoue, J. Sakai, and M. Edahiro. Processor virtualization for secure mobile terminals. *ACM Trans. on Des. Autom. Electron. Syst.*, 13(3):48:1–48:23, July 2008.

[156] M. Ismail. WiMAX: a competing or complementary technology to 3G? In *Proc. of the Conf. on Integrated Circuits and Syst. Design*, pages 3–3, Sept. 2007.

[157] V. Izosimov et al. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 864–869, Mar. 2005.

[158] X. Ji et al. Design and implementation of configurable simulation system for embedded software based on MDA. In *Proc. of the Summer Comp. Sim. Conf.*, pages 190–194, July 2010.

[159] Y. Jin and B. Sendhoff. Alleviating catastrophic forgetting via multi-objective learning. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 3335–3342, July 2006.

[160] N. P. Johnson et al. Fast condensation of the program dependence graph. In *Proc. of the Conf. on Programming Lang. Design and Impl.*, pages 39–50, 2013.

[161] K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments. *Info. Proc. and Management*, 36(6):779–808, Nov. 2000.

[162] Y. Joo, D. Lee, and Y. I. Eom. Delegating OpenGL commands to host for hardware support in virtualized environments. In *Proc. of the Int. Conf. on Ubiquitous Info. Management and Comm.*, pages 95:1–95:4, Jan. 2014.

[163] M. Jung et al. TLM modelling of 3D stacked wide I/O DRAM subsystems: A virtual platform for memory controller design space exploration. In *Proc. of the Workshop on Rapid Simulation and Perf. Eval.: Methods and Tools*, pages 5:1–5:6, Jan. 2013.

[164] Y. Jung and L. P. Carloni. ΣVP: Host-GPU multiplexing for efficient simulation of multiple embedded GPUs on virtual platforms. In *Proc. of the Design Automation Conf.*, pages 106:1–106:6, June 2015.

[165] Y. Jung et al. Altocumulus: Harvesting computational resources from devices at the edge of the cloud. Technical report, Columbia University, June 2013.

[166] Y. Jung, R. Neill, and L. P. Carloni. A broadband embedded computing system for MapReduce utilizing hadoop. In *Proc. of the Int. Conf. on Cloud Comp. Tech. and Science*, pages 1–9, Dec. 2012.

[167] Y. Jung, J. Park, M. Petracca, and L. P. Carloni. netShip: A networked virtual platform for large-scale heterogeneous distributed embedded systems. In *Proc. of the Design Automation Conf.*, pages 169:1–169:10, May 2013.

[168] Y. Jung, M. Petracca, and L. P. Carloni. Cloud-aided design for distributed embedded systems. *IEEE Design & Test*, 31(4):32–40, Jul-Aug 2014.

[169] Y. Jung, K. Stratos, and L. P. Carloni. LN-Annote: An alternative approach to information extraction from emails using locally-customized named-entity recognition. In *Proc. of the Int. World Wide Web Conf.*, pages 538–548, May 2015.

[170] S. Kågström, H. Grahn, and L. Lundberg. Cibyl: an environment for language diversity on mobile devices. In *Proc. of the Int. Conf. on Virtual execution environments*, pages 75–82, June 2007.

[171] T. Kakantousis et al. Misco: A system for data analysis applications on networks of smartphones using MapReduce. In *Proc. of the Int. Conf. on Mobile Data Management*, pages 356–359, July 2012.

[172] T. Kakimoto et al. Performance comparison of GPU programming frameworks with the striped smith-Waterman algorithm. *SIGARCH Comp. Arch. News*, 40(5):70–75, Mar. 2012.

[173] D.-I. Kang et al. A software synthesis tool for distributed embedded system design. In *Proc. of the Workshop on Languages, Compilers, and Tools for Embed. Syst.*, pages 87–95, May 1999.

[174] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. of the Symp. on Discrete Algorithms*, pages 938–948, Jan. 2010.

[175] J. Kato, H. Fujita, and Y. Ishikawa. Evaluation of energy-efficient cluster server using embedded processors. In *Proc. of the Int. Workshop on SW Tech. for Future Dependable Dist. Syst.*, pages 106–111, Mar. 2009.

[176] Y. Ke, D. Hoiem, and R. Sukthankar. Computer vision for music identification. In *Proc. of the Conf. on Comp. Vision and Pattern Recognition*, pages 597–604, June 2005.

[177] B. Kehoe et al. A survey of research on cloud robotics and automation. *Trans. on Automation Science and Engineering*, PP(99):1–12, Jan. 2015.

[178] T. Kempf et al. A SW performance estimation framework for early system-level-design using fine-grained instrumentation. In *Proc. of the Design, Automation, and Test in Europe Conf.*, pages 468–473, Mar. 2006.

[179] Y.-K. Ki. Accident detection system using image processing and MDR. *Int. Journal of Comp. Sci. and Net. Security*, pages 35–39, Mar. 2007.

[180] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk. Traffic characteristics of a massively multi-player online role playing game. In *Proc. of Workshop on Network and Syst. Support for Games*, pages 1–8, Oct. 2005.

[181] K.-H. Kim and S. Choi. Incremental learning to rank with partially-labeled data. In *Proc. of the Workshop on Web Search Click Data*, pages 20–27, Mar. 2009.

[182] K.-H. Kim, S.-J. Lee, and P. Congdon. On cloud-centric network architecture for multi-dimensional mobility. In *Proc. of the workshop on Mobile cloud comp.*, pages 1–6, Aug. 2012.

[183] Y. Kim and A. Shrivastava. Memory performance estimation of CUDA programs. *ACM Trans. on Embedded Comp.*, 13(2):21:1–21:22, Sept. 2013.

[184] M. Knaup and M. Kachelriess. Real-time CT image reconstruction using a mercury's DCBS and a sony's PS3 cluster. In *Nuclear Science Symposium Conf. Record*, volume 5, pages 3926–3928, Nov. 2007.

[185] A. Kobsa, B. P. Knijnenburg, and B. Livshits. Let's do it at my place instead?: Attitudinal and behavioral study of privacy in client-side personalization. In *Proc. of the Conf. on Human Factors in Comp. Syst.*, pages 81–90, Apr. 2014.

[186] T. Kogel and M. Braun. Virtual prototyping of embedded platforms for wireless and multimedia. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 488–490, 2006.

[187] P. Kollig, C. Osborne, and T. Henriksson. Heterogeneous multi-core platform for consumer multimedia applications. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 1254–1259, Apr. 2009.

[188] A. J. Komecki et al. Evaluation of software development tools for high assurance safety critical systems. In *Proc. of the Int. Conf. on High Assurance Syst. Engineering*, pages 273–274, Mar. 2004.

[189] G. Kotsis. 10 years mobile multimedia: from Motorola RAZR to iPhone 5. In *Proc. of the Int. Conf. on Inf. Integration and Web-based Apps. & Services*, pages 1–1, Dec. 2012.

[190] D. Koutsonikolas and Y. C. Hu. On the feasibility of bandwidth estimation in wireless access networks. *Wireless Net.*, 17(6):1561–1580, Aug. 2011.

[191] J. Kozhipurath. Cloud service costing challenges. In *Proc. of the Int. Conf on Cloud Comp. in Emerging Markets*, pages 1–6, Oct. 2012.

[192] H. Kruppa, M. CastrillonSantana, and B. Schiele. Fast and robust face finding via local context. In *Proc. of the Int. Workshop on VS-PETS*, pages 157–164, Oct. 2003.

[193] J. Kruse et al. Introducing flexible quantity contracts into distributed SoC and embedded system design processes. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 938–943, Mar. 2005.

[194] A. Kumar and E. Pilli. University wide m-learning using cloud environment. In *Proc. of the Int. Symp. on Cloud and Services Comp.*, pages 118–123, Dec. 2012.

[195] D. R. Kumar. and S. ManjupPriya. Cloud based m-healthcare emergency using SPOC. In *Proc. of the Int. Conf. on Advanced Comp.*, pages 286–292, Dec. 2013.

[196] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, Apr. 2010.

[197] J. Lai and A. Seznec. Break down GPU execution time with an analytical method. In *Proc. of the Workshop on Rapid Sim. and Perf. Eval.*, pages 33–39, Jan. 2012.

[198] O. Le Goaer and S. Waltham. Yet another DSL for cross-platforms mobile development. In *Proc. of the Workshop on the Globalization of Domain Specific Languages*, pages 28–33, July 2013.

[199] S.-M. Lee et al. Fine-grained I/O access control of the mobile devices based on the xen architecture. In *Proc. of the Int. Conf. on Mobile Comp. and Net.*, pages 273–284, Sept. 2009.

[200] P. Levis et al. TOSSIM: Accurate and scalable simulation of entire tinyOS applications. In *Proc. of the Int. Conf. on Embedded Net. Sensor Syst.*, pages 126–137, Nov. 2003.

[201] C. Li et al. TwiNER: Named entity recognition in targeted twitter stream. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 721–730, Aug. 2012.

[202] Q. Li and H. Ji. Incremental joint extraction of entity mentions and relations. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 402–412, June 2014.

[203] W. Li et al. Crowd density estimation: An improved approach. In *Int. Conf. on Signal Proc.*, pages 1213–1216, Oct. 2010.

[204] W. Li, Y. Liu, and X. Xue. Robust audio identification for MP3 popular music. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 627–634, July 2010.

[205] X. Li and G. Autran. Implementing an mobile agent platform for m-commerce. In *Proc. of the Int. Computer SW and App. Conf.*, volume 2, pages 40–45, July 2009.

[206] S. Lie et al. Design and implementation of hardware-in-the-loop-simulation for UAV using PID control method. In *Proc. of the Int. Conf. on Instrumentation, Comm., Info. Tech., and Biomedical Engineering*, pages 124–130, Nov. 2013.

[207] D. Liu and R. Deters. The reverse C10K problem for server-side mashups. In *Workshops on Service-Oriented Computing*, volume 5472 of *Lecture Notes in Computer Science*, pages 166–177. 2009.

[208] F. Liu et al. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Comm.*, 20(3):14–22, June 2013.

[209] T. Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Info. Retrieval*, (3):225–331, 2009.

[210] Y. Liu et al. Evolving SaaS based on reflective petri nets. In *Proc. of the Workshop on Reflection, AOP and Meta-Data for SW Evolution*, pages 7:1–7:4, June 2010.

[211] D. Loi. TV reinvented: designing pleasurable interfaces for the living room. In *Proc. of the Conf. on Designing Pleasurable Products and Interfaces*, pages 18:1–18:4, June 2011.

[212] M. Lombardi, M. Milano, and L. Benini. Robust non-preemptive hard real-time scheduling for clustered multicore platforms. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 803–808, Apr. 2009.

[213] N. E. Lownes and R. B. Machemehl. VISSIM: A multi-parameter sensitivity analysis. In *Proc. of the Winter Sim. Conf.*, pages 1406–1413, Dec. 2006.

[214] Y. Ma, J. Rao, W. Hu, X. Meng, X. Han, Y. Zhang, Y. Chai, and C. Liu. An efficient index for massive IOT data in cloud environment. In *Proc. of the Int. Conf. on Information and Knowledge Management*, pages 2129–2133, Oct. 2012.

[215] R. Mahindra et al. A practical traffic management system for integrated LTE-WiFi networks. In *Proc. of the Int. Conf. on Mobile Comp. and Networking*, pages 189–200, Sept. 2014.

[216] A. Marana et al. Estimating crowd density with Minkowski fractal dimension. In *Proc. of ICASSP*, volume 6, pages 3521–3524, Mar. 1999.

[217] D. McIntire et al. Energy-efficient sensing with the low power, energy aware processing (leap) architecture. *ACM Trans. on Embedded Comp. Syst.*, 11(2):27:1–27:36, July 2012.

[218] S. C. McLoone, P. J. Walsh, and T. E. Ward. An enhanced dead reckoning model for physics-aware multiplayer computer games. In *Proc. of the Int. Symp. on Dist. Sim. and Real Time App.*, pages 111–117, May 2012.

[219] L. McVoy and C. Staelin. lmbench: portable tools for performance analysis. In *Proc. of the Annual Conf. on USENIX Annual Tech. Conf.*, pages 279–284, Jan. 1996.

[220] F. P. Miller, A. F. Vandome, and J. McBrewster. *BD-J*. 2009.

[221] E. Miluzzo et al. Vision: mClouds - computing on clouds of mobile devices. In *Proc. of the Workshop on Mobile Cloud Comp. and Services*, pages 9–14, June 2012.

[222] C. Min et al. SFS: Random write considered harmful in solid state drives. In *Proc. of the Conf. on File and Storage Tech.*, Feb. 2012.

[223] S. M. Mitchell and C. B. Seaman. A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. In *Proc. of the Int. Symp. on Empirical Software Engineering and Measurement*, pages 511–515, Oct. 2009.

[224] D. Mitrovic, M. Zeppelzauer, and C. Breiteneder. Features for content-based audio retrieval. *Advances in Comp.: Improving the Web*, pages 71–150, Mar. 2010.

[225] A. Mohammad, K. Mohiuddin, M. Irfan, and M. Moizuddin. Cloud the mainstay: Growth of social networks in mobile environment. In *Proc. of the Int. Conf. on Cloud Ubiquitous Comp. Emerging Tech.*, pages 14–19, Nov. 2013.

[226] K. Mohiuddin et al. 24x7x365: Mobile cloud access. In *Proc. of the Int. Info. Tech. Conf.*, pages 544–551, Sept. 2012.

[227] P. R. L. Monica. 5 reasons to not 'like' Facebook's IPO. *CNN Money*, pages 1–1, May 2012.

[228] G. Moore. Cramming more components onto integrated circuits. *Proc. of the IEEE*, 86(1):82–85, Jan. 1998.

[229] C. Moretti, K. Steinhaeuser, D. Thain, and N. Chawla. Scaling up classifiers to cloud computers. In *Proc. of the Int. Conf. on Data Mining*, pages 472–481, Dec. 2008.

[230] D. Mueller-Gritschneder et al. A virtual prototyping platform for real-time systems with a case study for a two-wheeled robot. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 1331–1334, Mar. 2013.

[231] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Proc. of the Int. Conf. on Advances in Artificial Intelligence*, pages 266–277, Dec. 2006.

[232] J. Nagle. Congestion control in IP/TCP internetworks. *SIGCOMM Comp. Comm. Review*, 14(4):11–17, Oct. 1984.

[233] R. Neill et al. Embedded processor virtualization for broadband grid computing. In *Proc. of the IEEE/ACM Int. Conf. on Grid Comp.*, pages 145–156, Sept. 2011.

[234] R. Neill, A. Shabarshin, and L. P. Carloni. A heterogeneous parallel system running open mpi on a broadband network of embedded set-top devices. In *Proc. of the ACM Int. Conf. on Comp. Frontiers*, pages 187–196, May 2010.

[235] N. Ng, N. Yoshida, and K. Honda. Safe parallel programming with message optimisation. In *Proc. of Int. Conf. on Objects, Models, Components, Patterns*, volume 7304, pages 202–218, May 2012.

[236] H. M. Nguyen et al. An alternative approach to avoid overfitting for surrogate models. In *Proc. of the Winter Sim. Conf.*, pages 2765–2776, Dec. 2011.

[237] D. Nurmi et al. The Eucalyptus open-source cloud-computing system. In *Proc. of the Int. Symp. on Cluster Comp. and the Grid*, pages 124–131, May 2009.

[238] E. Oliver. The challenges in large-scale smartphone user studies. In *Proc. of the Int. Workshop on Hot Topics in Planet-scale Measurement*, pages 5:1–5:5, June 2010.

[239] E. Ordoñez Cardenas and R. d. J. Romero-Troncoso. MLP neural network and on-line back-propagation learning implementation in a low-cost FPGA. In *Proc. of the Symp. on VLSI*, pages 333–338, May 2008.

[240] S. Orlando, F. Pizzolon, and G. Tolomei. SEED: A framework for extracting social events from press news. In *Proc. of the Int. Conf. on World Wide Web Companion*, pages 1285–1294, May 2013.

[241] Z. Ou et al. Energy- and cost-efficiency analysis of ARM-based clusters. In *Proc. of the Int. Symp. on Cluster, Cloud and Grid Comp.*, pages 115–123, May 2012.

[242] J. Owens et al. GPU computing. *Proc. of the IEEE*, 96(5):879–899, May 2008.

[243] W. Paik et al. Applying natural language processing (NLP) based metadata extraction to automatically acquire user preferences. In *Proc. of the Int. Conf. on Knowledge Capture*, pages 116–122, Oct. 2001.

[244] C. Parikh and P. Patel. Performance evaluation of AES algorithm on various development platforms. In *Proc. of the Int. Conf. on Consumer Electronics*, pages 1–6, June 2007.

[245] S. I. Park, V. Raghunathan, and M. B. Srivastava. Energy efficiency and fairness tradeoffs in multi-resource, multi-tasking embedded systems. In *Proc. of the Int. Symp. on Low power electronics and design*, pages 469–474, Aug. 2003.

[246] C. Partridge. Realizing the future of wireless data communications. *Comm. of the ACM*, 54(9):62–68, Sept. 2011.

[247] S. Pawar, R. Srivastava, and G. K. Palshikar. Automatic gazette creation for named entity recognition and application to resume processing. In *Proc. of the Conf. on Intelligent & Scalable Syst. Tech.*, pages 15:1–15:7, Jan. 2012.

[248] R. Picek and M. Grcic. Evaluation of the potential use of m-learning in higher education. In *Proc. of the Int. Conf. on Info. Tech. Interfaces*, pages 63–68, June 2013.

[249] G. Pierre and C. Stratan. ConPaaS: A platform for hosting elastic cloud applications. *IEEE Internet Computing Magazine*, 16(5):88–92, Sept. 2012.

[250] R. Pinilla and M. Gil. ULT: a Java threads model for platform independent execution. *SIGOPS Operating Syst. Review*, 37(4):48–62, Oct. 2003.

[251] A. Pintus, D. Carboni, and A. Piras. The anatomy of a large scale social web for internet enabled objects. In *Proc. of the Int. Workshop on Web of Things*, pages 6:1–6:6, June 2011.

[252] R. Polikar et al. Learn++: an incremental learning algorithm for supervised neural networks. *Trans. on Syst., Man, and Cybernetics*, 31(4):497–508, Nov. 2001.

[253] T. Pop, P. Eles, and Z. Peng. Design optimization of mixed time/event-triggered distributed embedded systems. In *Proc. of the Int. Conf. on HW/SW Codesign and Syst. Synthesis*, pages 83–89, Sept. 2003.

[254] A. Poritz. Hidden Markov models: a guided tour. In *Proc. of the Int. Conf. on Acoustics, Speech, and Signal Proc.*, pages 7–13, Apr. 1988.

[255] V. Puranik, T. Mitra, and Y. N. Srikant. Probabilistic modeling of data cache behavior. In *Proc. of the Int. Conf. on Embedded Software*, pages 255–264, Oct. 2009.

[256] D. Quaglia et al. Timing aspects in QEMU/SystemC synchronization. *Proc. of the Int. QEMU Users' Forum*, pages 11–14, Mar. 2011.

[257] Qualcomm Technologies Inc. Trepn: Profiler starter edition user guide. In *Qualcomm Developer Network*, pages 1–46. Apr. 2014.

[258] S. Radicati. Mobile statistics report, 2014-2018. *The Radicati Group*, pages 1–3, Feb. 2014.

[259] A. Rahmati et al. Exploring iPhone usage: the influence of socioeconomic differences on smartphone adoption, usage and usability. In *Proc. of the Int. Conf. on HCI with mobile devices and services*, pages 11–20, Sept. 2012.

[260] H. Raj et al. Enabling semantic communications for virtual machines via iConnect. In *Proc. of the Int. Workshop on Virtualization Tech. in Dist. Comp.*, pages 1:1–1:8, Nov. 2007.

[261] V. Ramharuk and I. Osunmakinde. Cloud robotics: A framework towards cloud-enabled multi-robotics survivability. In *Proc. of the Conf. on Empowered by Tech.*, pages 82:82–82:92, Oct. 2014.

[262] V. T. Ravi et al. Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework. In *Proc. of the Int. Symp. on High-Perf. Parallel and Dist. Comp.*, pages 217–228, June 2011.

[263] F. J. Ridruejo et al. Concepts and components of full-system simulation of distributed memory parallel computers. In *Proc. of the Int. Symp. on High Perf. Dist. Comp.*, pages 225–226, 2007.

[264] T. Ristenpart et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, pages 199–212, Nov. 2009.

[265] A. Ritter et al. Modeling missing data in distant supervision for information extraction. *Trans. of the Association for Computational Linguistics*, pages 367–378, 2013.

[266] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Info. Retrieval*, 3(4):333–389, Apr. 2009.

[267] A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Proc. of the Int. Conf. on Artificial Neural Networks and Expert Syst.*, pages 65–68, Nov. 1993.

[268] S. Roche and A. Rajabifard. Sensing places' life to make city smarter. In *Proc. of the Int. Workshop on Urban Comp.*, pages 41–46, Aug. 2012.

[269] T. Roelleke and J. Wang. TF-IDF uncovered: A study of theories and probabilities. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 435–442, July 2008.

[270] H. Roitman et al. Harnessing the crowds for smart city sensing. In *Proc. of the Int. Sorkshop on Multimodal Crowd Sensing*, pages 17–18, Nov. 2012.

[271] K. Rust. Using little's law to estimate cycle time and cost. In *Proc. of the Conf. on Winter Sim.*, pages 2223–2228, Dec. 2008.

[272] Z. Sanaei et al. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *Communications Surveys Tutorials*, 16(1):369–392, May 2013.

[273] M. C. Santana et al. Face and facial feature detection evaluation. In *Proc. of VISAPP*, pages 167–172, Apr. 2008.

[274] G. Saon et al. Speaker adaptation of neural network acoustic models using i-vectors. In *Proc. of the Workshop on Automatic Speech Recognition and Understanding*, pages 55–59, Dec. 2013.

[275] E. Sarigol, D. Garcia, and F. Schweitzer. Online privacy as a collective phenomenon. In *Proc. of the Conf. on Online Social Networks*, pages 95–106, Oct. 2014.

[276] P. Sayyah et al. Virtual platform-based design space exploration of power-efficient distributed embedded applications. *Trans. on Embeded Comp. Syst.*, 14(3):49:1–49:25, Apr. 2015.

[277] C. Scharfenberger, S. Chakraborty, and G. Färber. Robust image processing for an omnidirectional camera-based smart car door. *Trans. on Embedded Comp. Syst.*, 11(4):87:1–87:28, Jan. 2013.

[278] M. Schoeberl, S. Korsholm, T. Kalibera, and A. P. Ravn. A hardware abstraction layer in Java. *Trans. on Embedded Comp. Syst.*, 10(4):42:1–42:40, Nov. 2011.

[279] M. J. Sebern. Iterative development and commercial tools in an undergraduate software engineering course. *SIGCSE Bulletin*, 29(1):306–309, Mar. 1997.

[280] S. Seneviratne et al. Personal cloudlets for privacy and resource efficiency in mobile in-app advertising. In *Proc. of the Int. Workshop on Mobile Cloud Comp. and Net.*, pages 33–40, July 2013.

[281] D. E. Setliff, J. K. Strosnider, and J. A. Madriz. Towards a design assistant for distributed embedded systems. In *Proc. of the Int. Conf. on ASE*, pages 311–312, Nov. 1997.

[282] M. Shand, P. Bertin, and J. Vuillemin. Hardware speedups in long integer multiplication. *SIGARCH Comp. Arch. News*, 19(1):106–113, Mar. 1991.

[283] J. W. Sheaffer, D. Luebke, and K. Skadron. A flexible simulation framework for graphics architectures. In *Proc. of the Conf. on Graphics Hardware*, pages 85–94, Aug. 2004.

[284] G. Shen et al. Accelerate video decoding with generic GPU. *Trans. on Circuits and Syst. for Video Tech.*, 15(5):685–693, May 2005.

[285] D. Shey. Mobile cloud computing reaches $5.2b market by 2015. *ABI Research: Enterprise Mobile Cloud Computing*, pages 1–98, Dec. 2009.

[286] C. Shi et al. COSMOS: Computation offloading as a service for mobile devices. In *Proc. of the Int. Symp. on Mobile Ad Hoc Networking and Comp.*, pages 287–296, Aug. 2014.

[287] T. Shi, R. Wang, D. Zhang, W. Jiao, and B. Xie. The analysis of the behavior patterns of components (intelligent sensors) in the IoT-oriented internetware. In *Proc. of the Asia-Pacific Symp. on Internetware*, pages 22:1–22:4, Oct. 2013.

[288] T. Shimokawabe et al. An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code. In *Proc. of the Int. Conf. for High Perf. Comp., Net., Storage and Anal.*, pages 1–11, Nov. 2010.

[289] E. Shin et al. Valuating the economic impacts of urban environmental problems: Asian cities. *Urban Management and Poverty Reduction*, (30073):1–119, June 1997.

[290] R. C. Singleton. On computing the fast Fourier transform. *Comm. of the ACM*, 10(10):647–654, Oct. 1967.

[291] J. Slawinski and V. Sunderam. Adapting MPI to MapReduce PaaS clouds: An experiment in cross-paradigm execution. In *Proc. of the Int. Conf. on Utility and Cloud Comp.*, pages 199–203, Nov. 2012.

[292] J. Sommers and P. Barford. Cell vs. WiFi: On the performance of metro area mobile connections. In *Proc. of the Internet Measurement Conf.*, pages 301–314, Nov. 2012.

[293] C. Staelin and L. McVoy. mhz: anatomy of a micro-benchmark. In *Proc. of the Annual Conf. on USENIX Annual Tech. Conf.*, pages 155–166, June 1998.

[294] A. Stanik, M. Hovestadt, and O. Kao. Hardware as a Service (HaaS): The completion of the cloud stack. In *Proc. of the Int. Conf. on Comp. Tech. and Info. Management*, volume 2, pages 830–835, Apr. 2012.

[295] D. Stowell and M. D. Plumbley. An open dataset for research on audio field recording archives: freefield1010. In *Proc. of the Int. Conf. on Semantic Audio*, Jan. 2014.

[296] R. Stratonovich. Conditional markov processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960.

[297] D. Su et al. SmartVisor: towards an efficient and compatible virtualization platform for embedded system. In *Proc. of the Workshop on Isolation and Integration in Embedded Syst.*, pages 37–41, Mar. 2009.

[298] S. Sud et al. Dynamic migration of computation through virtualization of the mobile platform. *Journal of Mobile Net. App.*, 17(2):206–215, Apr. 2012.

[299] T. H. Szymanski. Low latency energy efficient communications in global-scale cloud computing systems. In *Proc. of the Workshop on Energy Efficient High Perf. Parallel and Dist. Comp.*, pages 13–22, June 2013.

[300] T. Taha et al. Neuromorphic algorithms on clusters of PlayStation 3s. In *Proc. of the Int. Joint Conf. on Neural Net.*, pages 1–10, July 2010.

[301] K. Tang, J. M. Zhang, and Z. B. Jiang. Framework for SaaS management platform. In *Proc. of the Int. Conf. on e-Business Engineering*, pages 345–350, Nov. 2010.

[302] D. Taylor. Need for speed: PS3 Linux! *Linux Journal*, (156):5–6, Apr. 2007.

[303] J. Tejedor et al. Comparison of methods for language-dependent and language-independent query-by-example spoken term detection. *Trans. on Info. Syst.*, 30(3):18:1–18:34, Sept. 2012.

[304] N. Thanh-Cuong, S. Wen-Feng, C. Ya-Hui, and X. Wei-Min. Research and implementation of scalable parallel computing based on map-reduce. *Journal of Shanghai Univ.*, 15(5):426–429, Aug. 2011.

[305] The Guardian. Naked celebrity hack: security experts focus on iCloud backup theory, Sept. 2014.

[306] The Washington Post. 4chan: The 'shock post' site that hosted the private Jennifer Lawrence photos, Sept. 2014.

[307] The Washington Post. Proposed prince william data center prompts protest letter to Jeff Bezos, Jan. 2015.

[308] A. Tiwana. Impact of classes of development coordination tools on software development performance: A multinational empirical study. *Trans. on Software Eng. Methodology.*, 17(2):11:1–11:47, May 2008.

[309] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of the Conf. on Natural Lang. Learning*, pages 142–147, May 2003.

[310] M. Tkachenko and A. Simanovsky. Named entity recognition: exploring features. In *Proc. of the Conf. on Natural Lang. Proc.*, pages 118–127, Sept. 2012.

[311] V. R. Tomás and L. A. Garcia. A cooperative multiagent system for traffic management and control. In *Proc. of the Int. Conf. on Autonomous Agents and Multiagent Syst.*, pages 52–59, July 2005.

[312] V. Toubiana, V. Verdot, and B. Christophe. Cookie-based privacy issues on Google services. In *Proc. of the Conf. on Data and App. Security and Privacy*, pages 141–148, Feb. 2012.

[313] J. Trajkovic, A. V. Veidenbaum, and A. Kejariwal. Improving sdram access energy efficiency for low-power embedded systems. *ACM Trans. on Embedded Comp. Syst.*, 7(3):24:1–24:21, May 2008.

[314] S. S. Tsai, D. Chen, J. P. Singh, and B. Girod. Rate-efficient, real-time cd cover recognition on a camera-phone. In *Proc. of the ACM Intl. Conf. on Multimedia*, pages 1023–1024, Oct. 2008.

[315] W.-T. Tsai, Y. Huang, and Q. Shao. EasySaaS: A SaaS development framework. In *Proc. of the Int. Conf. on Service-Oriented Comp. and App.*, pages 1–4, Dec. 2011.

[316] J. Tsujii. Generic NLP technologies: Language, knowledge and information extraction. In *Proc. of the Annual Meeting on Assoc. for Comp. Linguistics*, pages 12–22, Oct. 2000.

[317] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 384–394, July 2010.

[318] R. Ubal et al. Multi2Sim: A simulation framework for CPU-GPU computing. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 335–344, Sept. 2012.

[319] S. N. V, P. Mitra, and S. K. Ghosh. Conditional random field based named entity recognition in geological text. *Int. Journal of Comp. Applications*, pages 119–122, 2010.

[320] C. H. K. van Berkel. Multi-core for mobile phones. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 1260–1265, Mar. 2009.

[321] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, Nov. 2008.

[322] K. Vandenbroucke et al. Mobile cloud storage: A contextual experience. In *Proc. of the Int. Conf. on Human-computer Interaction with Mobile Devices and Services*, pages 101–110, Sept. 2014.

[323] A. Velivelli, C. Zhai, and T. Huang. Audio segment retrieval using a short duration example query. In *Proc. of the Int. Conf. on Multimedia and Expo*, volume 3, pages 1603–1606, June 2004.

[324] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *Proc. of ICCV*, volume 2, pages 734–741, Oct. 2003.

[325] D. T. Wagner et al. Device analyzer: Understanding smartphone usage. In *Proc. of the Int. Conf. on Mobile and Ubiq. Syst.: Comp., Net., and Services*, pages 195–208, Dec. 2014.

[326] A. L.-C. Wang. An industrial-strength audio search algorithm. In *Proc. of the Int. Conf. on Music Info. Retrieval*, pages 713–718, Oct. 2003.

[327] C. Wang and Z. Li. Parametric analysis for adaptive computation offloading. *SIGPLAN Note*, 39(6):119–130, June 2004.

[328] C.-C. Wang et al. NetVP: A system-level network virtual platform for network accelerator development. In *IEEE Int. Symp. on Circuits and Systems*, pages 249–252, May 2012.

[329] J. H. Wang and H. Y. Wang. Incremental neural network construction for text classification. In *Proc. of the Int. Symp. on Comp., Consumer and Control*, pages 970–973, June 2014.

[330] L. Wang et al. Exploring the influencing factors on surfing behavior intention of smartphone users. In *Proc. of the Int. Workshop on Adv. Comp. Intelligence*, pages 688–692, Oct. 2011.

[331] S. Wang, Y. Liu, and S. Dey. Wireless network aware cloud scheduler for scalable cloud mobile gaming. In *Proc. of the Int. Conf. on Comm.*, pages 2081–2086, June 2012.

[332] X. Wang et al. SHIP: Scalable hierarchical power control for large-scale data centers. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 91–100, Sept. 2009.

[333] X. Wang et al. AppMobiCloud: Improving mobile web applications by mobile-cloud convergence. In *Proc. of the Asia-Pacific Symp. on Internetware*, pages 14:1–14:10, Oct. 2013.

[334] Y. Wang, X. Lin, and M. Pedram. A nested two stage game-based optimization framework in mobile cloud computing system. In *Proc. of the Int. Symp. on Service Oriented Syst. Eng.*, pages 494–502, Mar. 2013.

[335] Y. Wang, J. Wu, and W.-S. Yang. Cloud-based multicasting with feedback in mobile social networks. *IEEE Trans. on Wireless Comm.*, 12(12):6043–6053, Dec. 2013.

[336] Y.-C. Wang, B. Donyanavard, and K.-T. T. Cheng. Energy-aware real-time face recognition system on mobile CPU-GPU platform. In *Proc. of the European Conf. on Trends and Topics in Comp. Vision*, volume 2, pages 411–422, Sept. 2012.

[337] Y.-C. Wang, S. Pang, and K.-T. Cheng. A GPU-accelerated face annotation system for smartphones. In *Proc. of the Multimedia Conf.*, pages 1667–1668, Oct. 2010.

[338] Z. Wang et al. SysCOLA: A framework for co-development of automotive software and system platform. In *Proc. of the Design Automation Conf.*, pages 37–42, July 2009.

[339] M. Wenzel. Shared-memory multiprocessing for interactive theorem proving. In *Proc. of the Int. Conf. on Interactive Theorem Proving*, pages 418–434, July 2013.

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

[341] S. Whittaker, V. Bellotti, and J. Gwizdka. Email in personal information management. *Comm. of the ACM*, 49(1):68–73, Jan. 2006.

[342] A. Wolisz. Desirable trends in mobile communication. *SIGMOBILE Mob. Comp. Comm. Rev.*, 16(1):2–9, July 2012.

[343] H. Wong et al. Demystifying GPU microarchitecture through microbenchmarking. In *Proc. of the Int. Symp. on Perf. Analysis of Syst. and Software*, pages 235–246, Mar. 2010.

[344] Y. Xiangzhan et al. Research on performance estimation model of distributed network simulation based on PDNS conservative synchronization mechanism in complex environment. In *Proc. of the Int. Conf. on Comp. and Info. Tech.*, pages 2576–2580, Dec. 2010.

[345] R. Xu et al. Energy-efficient policies for embedded clusters. *SIGPLAN Not.*, 40(7):1–10, June 2005.

[346] H. Yan, X. Zhang, and H. Xu. Three levels intelligent incident detection algorithm of smart traffic in the digital city. In *Proc. of the Int. Conf. on Adv. Intelligent Comp. Theories and App.: With Aspects of AI*, pages 260–266, Aug. 2011.

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

[348] L. Yang et al. A framework for partitioning and execution of data stream applications in mobile cloud computing. *SIGMETRICS Perf. Evaluation Review*, 40(4):23–32, Apr. 2013.

[349] Y. Yang et al. Shared memory multiplexing: A novel way to improve GPGPU throughput. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 283–292, Sept. 2012.

[350] S. R. Yerva et al. Social and sensor data fusion in the cloud. In *Proc. of the Int. Conf. on Mobile Data Management*, pages 276–277, July 2012.

[351] B. Yu et al. On managing very large sensor-network data using bigtable. In *Proc. of the Int. Symp. on Cluster, Cloud and Grid Comp.*, pages 918–922, May 2012.

[352] T. Yu, , et al. SimRacer: An automated framework to support testing for process-level races. In *Proc. of the Int. Symp. on Software Testing and Analysis*, pages 167–177, July 2013.

[353] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of the Int. Conf. on HW/SW Codesign and Syst. Synthesis*, pages 105–114, Oct. 2010.

[354] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of the Int. Conf. on Hardware/Software Codesign and Syst. Synthesis*, pages 105–114, Oct. 2010.

[355] J. Zhou, X. Lin, X. Dong, and Z. Cao. PSMPA: Patient self-controllable and multi-level privacy-preserving cooperative authentication in distributed m-healthcare cloud computing system. *IEEE Trans. on Parallel and Dist. Syst.*, PP(99):1–11, 2014.

[356] M. M. Zloof. Query by example. In *Proc. of the National Comp. Conf. and Exposition*, pages 431–438, May 1975.

[357] L. Zuolo et al. SSDExplorer: A virtual platform for fine-grained design space exploration of solid state drives. In *Proc. of the Conf. on Design, Automation & Test in Europe*, pages 284:1–284:6, Mar. 2014.

Part V

Appendix: Prototype Tools

Appendix: Prototype Tools

This appendix describes the following four prototype tools that I developed as part of my dissertation:

- NETSHIP: the networked virtual platform presented in Chapter 4 and 5. It includes management scripts, the synchronizer, and case study applications.
- *A Java Porting Tool*: a tool used in Chapter 7 that can be used to port Java applications to a different Java runtime stack. The tool includes the porting script and the class dependency analyzer.
- *LN-Annote*: a training algorithm for NER that can learn in two separated steps, first one in the cloud and another in the local device.
- *Slime*: a light-weighted management tool for distribute computing nodes. Slime is used in the audio stream retrieval system presented in Chapter 9.

Throughout the appendix, a tutorial example is provided for each tool.

1 NETSHIP

There are three tar balls released with NETSHIP. *netShip.bz2* is the main archive file that includes the main features of NETSHIP such as a synchronizer, scripts for management, and two case studies. *OVP.bz2* is the archive that includes ready-to-go OVP instances of virtual platforms with ARM processors while *QMEU.bz2* comprises QEMU instances of virtual platforms with PPC processors. Originally, there existed four OVP instances and eight QEMU instances in each configuration, but only one instance in each archive is preserved to reduce the size of archives.

1.1 Software Requirements

The following software and libraries are required for the execution of NETSHIP. The version number enclosed by the parentheses denotes the software/library version that has been tested with NETSHIP.

- **Linux**: Linux kernel [https://www.kernel.org] (2.6.38-13)
- **gcc**: GNU Compiler Collection [https://gcc.gnu.org] (4.5.2)
- **tc**: Traffic Control [https://www.kernel.org/pub/linux/utils/net/iproute2] (20100519-3)
- **OVP**: Open Virtual Platform [http://imperas.com] (20111125)
- **QEMU**: Quick Emulator [http://www.qemu.org] (0.14.1)
- **Java**: Java SDK [http://www.java.com] (1.6.0 Update 22)
- **Eclipse**: Eclipse Java IDE [https://eclipse.org] (4.2)
- **Android**: Android SDK [https://developer.android.com] (r20.0.3)
- **CUDA SDK**: NVIDIA CUDA SDK [https://developer.nvidia.com/cuda-downloads] (5.5)

1.2 File Structure

The *netShip.bz2* file includes the following directories.

- **cs1/** directory for Case Study I.
- **cs1/mpi/** OpenMPI ported to the ARM and PPC processors.
- **cs1/scheduler/** OpenMPI scheduler based on the Simplex algorithm.
- **cs1/simplex/** a library to solve Simplex problems.
- **cs2/** directory for Case Study II.
- **cs2/CamSimCloudImgFetcher/** an Android app that fetches geo-tagged images from Flickr and Picassa as if they were taken through the camera installed to the Android device.
- **cs2/mappaint/** a tool to draw dots to a map using geo-location data tagged in the images.

- **cs2/opencv/** OpenCV library ported to PPC processors.
- **gpumulti/** a module that multiplexes the host GPU for the simulation of mobile GPUs.
- **guests/** drivers installed the virtual platform and applications that run on the guest virtual platform instances.
- **host/** configurations and applications that run on the host virtual machine instances.
- **host/synchronizer/** the synchronizer module that ensures the all the virtual platform instances proceed on the same simulation time with an adjustable time step granularity.
- **scripts/** scripts for the network and VP instance management.

OVP.bz2 and *QMEU.bz2* have the virtual platform instance configurations and the virtual hard disk images.

1.3 Tutorial

To setup NETSHIP, you need at least two virtual machine instances. One instance will be used as the main NETSHIP server while the rest will be the host virtual machines for the virtual platform instances. In NETSHIP, a host machine instance executes a same type of virtual platforms multiple times simultaneously to form a heterogeneous system.

1.3.1 Server Configuration and Setup

To configure the server instance, follow the instructions below:

1. Untar the *netShip.bz2* on a server computer that will serve as the main NETSHIP server where the synchronizer runs.
2. Launch *host/syncmaster*.

1.3.2 Setting Up a Host

To configure a host machine instance, follow the instructions below.

1. Create your own virtual machine instance using the virtual machine solution of your choice, e.g., VMWare for Desktop, VSphere, or AWS EC^2.

2. Change the NETSHIP server's IP address in *synchronizer/include/comm.h* and launch a VP Controller process (*synchronizer/vpctrl*) on the host machine instance (not inside the virtual platform).

3. Install the virtual platform instances on the host machine instance. You can decompress one of the given virtual platform instances, i.e., *OVP.bz2* or *QEMU.bz2*. Note that OVP requires an update of their license periodically.

4. Execute a VP Control Client process (*synchronizer/vpclient*) for each VP.

2 A Java Porting Tool

There are two tar balls released with this porting tool. *java_port.bz2* consists of the essential scripts and executables to port a Java program. *java_port_workspace.bz2* contains the entire workspace for porting Hadoop to an embedded Java stack.

2.1 Software Requirements

The following software and libraries are required for the execution of the porting tool. The version number enclosed by the parentheses denotes the software/library version that has been tested with the tool.

- **Linux**: Linux kernel [https://www.kernel.org] (2.6.32-5)
- **Java**: Java SDK [http://www.java.com] (1.6.0 Update 26-b03)

2.2 File Structure

The *java_port.bz2* includes the following directories.

- **port/** the main directory for the porting tool.

- **port/Retrotranslator/** the modified retrotranslator library that backports the input class files into older versions.
- **port/JavaProfileGapFiller/** implementation for the Java class files that are missing in the embedded Java stack, including *java.nio*, *java.nio.channels*, I/O stream classes, process management classes, and so on.
- **port/jdepend-2.9.1** an extended tool to analyze dependencies among Java class files.
- **port/run.sh** the main script that runs the entire porting process.

The *java_port_workspace.bz2* includes the following directories.

- **dependency/** the porting workspace.
- **usb/** the directory for files to be stored in a usb thumb drive. The files in this directory are used to boot the embedded devices and launch the start applet on the device.

2.3 Tutorial

The workspace archive file contains all the necessary files to port Hadoop 1.0.3 to an OCAP Java stack.

2.3.1 Porting Hadoop to the OCAP Stack

1. Untar *java_port_workspace.bz2*.
2. Go into the directory *dependency/converting/hadoop1_stb*
3. Edit the path variables in *run.sh*.
4. Execute the *run.sh* file.

3 LN-Annote

3.1 Software Requirements

The following software and libraries are required for the execution of LN-Annote. The version number enclosed by the parentheses denotes the software/library version that has been tested with

LN-Annote.

- **Linux**: Linux kernel [https://www.kernel.org] (3.2.0-77)
- **Java**: Java SDK [http://www.java.com] (1.6.0 Update 35)
- **Eclipse**: Eclipse Java IDE [https://eclipse.org] (4.2)
- **Android**: Android SDK [https://developer.android.com] (r21.0.1)
- **Android NDK**: Android Native Development Kit [https://developer.android.com/ndk] (r10c)
- **CUDA SDK**: NVIDIA CUDA SDK [https://developer.nvidia.com/cuda-downloads] (5.5)

3.2 File Structure

The bzip2 archive includes the following directories.

- **AndroidLnAnnote/** an Android container app that executes local training.
- **asset/** the OpenCL accelerated implementation for the training method.
- **c/** the Java Native Interface implementation that glues the OpenCL and Java code.
- **data/** the universal training and test data.
- **java/** the original Java implementation of the training method.
- **tsne/** a tool that reduces dimensions of input data and plots on a 2D space.

3.3 Tutorial

3.3.1 Executing the Java implementation

Executing the Java implementation is quite simple. This program will show the results of the training as well as the execution times.

1. Build the project by typing *ant*.
2. Execute the program by typing *ant run*.

3.3.2 Executing the OpenCL as an Android app

We use Eclipse to launch the Android app on the Android Emulator or on a real Android device. This program will show the results of the training as well as the execution times.

1. Create an Eclipse project using the directory *AndroidLnAnnote*.
2. Build the project using Eclipse.
3. Launch the built program on the Android Emulator or a connected Android device. If no debugging is necessary, you can just copy the apk file created to the device, install the file, and launch the program.

4 Slime

While most of the software used in the Audio Stream Retrieval system is proprietary, a portion of the system is released as an open source project. Slime is a light-weighted distribute computing framework that eases developing, deploying, and scaling of distributed computing nodes as well as event handling.

4.1 Software Requirements

The following software and libraries are required for the execution of Slime. The version number enclosed by the parentheses denotes the software/library version that has been tested with Slime.

- **Java**: Java SDK [http://www.java.com] (1.6.0 Update 35)

4.2 File Structure

The repository includes the following directories.

- **conf/** main configuration.
- **docs/** build restriction rules.
- **env/** configurations of libraries used, e.g., log4j.

- **lib/** libraries used by Slime.
- **src/** a source code directory.
- **tools/** build tools.

4.3 Tutorial

Slime can run either a stand alone program or be linked as a library.

4.3.1 Standalone Execution

Slime supports the plug-in architecture that loads plug-ins dynamically and runs their services. At this time this feature is still experimental. The way to launch Slime as a standalone process is as follows:

- java -cp dist/slime-1.0-20150225.jar:lib/commons-logging-1.1.1.jar:lib/grizzly-framework-2.3.5.jar: lib/grizzly-http-all-2.3.5.jar:lib/jersey-container-grizzly2-http-2.2.jar:lib/jsch-0.1.50.jar: lib/log4j-1.2.15.jar edu.columbia.slime.Slime

4.3.2 Using Slime as a Library

edu/columbia/slime/example/Example1.java is an example source code that runs as a process. It creates a Slime service and registers it to Slime. The Slime core logic dispatches the events its services requested. The following command will launch the example.

- java -cp dist/slime-1.0-20150225.jar:lib/commons-logging-1.1.1.jar:lib/grizzly-framework-2.3.5.jar: lib/grizzly-http-all-2.3.5.jar:lib/jersey-container-grizzly2-http-2.2.jar:lib/jsch-0.1.50.jar: lib/log4j-1.2.15.jar edu.columbia.slime.example.Example1