UNIVERSITY OF CALIFORNIA
Los Angeles

BigMobile: Data-Intensive Mobile Cloud Computing

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Jong Hoon Ahnn

2015

ABSTRACT OF THE DISSERTATION

BigMobile: Data-Intensive Mobile Cloud Computing

by

Jong Hoon Ahnn

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2015

Professor Miodrag Potkonjak, Chair

Mobile computing, where mobile users continuously gather, process, and share sensor or application-specific data, is emerging as a new computing and network paradigm of data sharing in seamless manner. The key enablers are the smartphones (e.g., iPhones and Android phones) equipped with onboard sensors (e.g., cameras, accelerometer, compass, GPS) and various wireless devices (e.g., WiFi, 3G/4G-LTE, and other network standards). However, despite of all the advances in recent years, mobile devices have limited resources for computation, memory, network, and battery. Mobile cloud computing (MCC) is a promising practical approach to relax such constraints in mobile devices for computational intensive applications, where computation offloading to virtually unlimited resources in clouds can help them to save the energy consumption.

Existing offloading algorithms and corresponding MCC systems offload computationally intensive applications to remote cloud servers in a distributed manner. However, they fail to address challenges in modern *data-intensive* applications, where sensor data is continuously harvested and application-specific data among users is shared to process or personalize applications. For instance, modern machine learning (ML)-based applications require the large amount of training data to train a model, often require periodic retraining or adaptation based on gathered user's specific profiles and context data. For such applications, offloading is not only computationally intensive but data intensive, and it requires a seamless interface to upload/fetch data to/from cloud

storage.

A comprehensive survey provides 15 different perspectives in ten major MCC systems from the viewpoint of data-intensive computation offloading. Typical MCC surveys often constitute network protocol, code rewrite requirement, offloading granularity, profiling, resource monitoring, cost model, and software preinstallation requirement. Our survey differs from typical MCC surveys mainly in several ways: scalable big data and computing support and fault-tolerance of MCC systems, providing features such as parallel offloading, multi-cloud support, disconnected operation, reliable message delivery, cloud sever scalability, and cloud storage access.

The process of computation offloading is complex. In this thesis, we tackle it as several subsystem problems: First, existing MCC systems fail to address multiple cloud settings in computation offloading scenarios. The application partitioning and offloading to multiple clouds are formally formulated and solved as an integer linear programming (ILP) problem. The objective function of the partitioning problem makes a set of offloading decision in terms of tradeoffs between mobile execution and cloud execution with respect to energy costs and time cost under the assumption of multi-clouds; Second, existing MCC systems fail to address a wireless network instability problem. In order to achieve fault-tolerant communication between mobile devices and cloud servers, we propose and implement a mechanism that can tolerate unstable network conditions by asynchronous implementation of network binding; Third, existing MCC systems fail to achieve reliable offloading message delivery, where millions of mobile devices submit computation offloading requests and data related requests to clouds. To achieve such scalability and reliability, we propose a producer/consumer-based message queue, where producers for delivering offloading requests send offloading requests over the network to the cluster, which in turn serves them up to consumers. The multiple consumers are in charge of submitting the offloaded tasks to cloud resources. Eventual reliability is achieved by message partitioning and replication in the message queue system while scalable message delivery is achieved by horizontal scaling of produc-

ers/consumers and reconfiguring of message partition factor. Fourth, cloud resource management is another challenging area, where researchers put little attention to data intensive application offloading. Existing MCC systems are not scalable when dealing with millions of offloading requests as well as data requests. For the data intensive application offloading, we consider integration to one of the conventional big data systems where a thin mobile client is to access data and to process it from the cloud. We propose the fuse of Hadoop-based parallel computation offloading to multiple clouds by developing seamless interfaces. The integration enables our MCC system to be scalable and reliable in scheduling offloading tasks and in serving data-related operations on top of Hadoop.

We provide a couple of evaluation results in terms of time saving and energy saving based on two types of benchmarks: computationally-intensive applications and data-intensive applications. For the former, we consider chess game, puzzle game, cryptographic algorithm, and Huffman compression algorithm, for the latter, we consider the state-of-the-art speech recognition application. The evaluation extends a single offloading to parallel offloading. The further evaluation of data-intensive computational cloud is performed in several aspects: performance comparison of Hadoop version 1 and version 2, Hadoop overhead, benefit of file compression, and performance of cloud storage to support small/large files. Finally, the performance of the proposed message queue is presented.

The dissertation of Jong Hoon Ahnn is approved.

Greg Pottie

Leonard Kleinrock

Mario Gerla

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2015

Thanks to God!

To my wife Yeonsun,

my son Seungjoo,

and my family in South Korea

from the bottom of my heart.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

VITA

1997–2004	B.S, Electrical and Computer Engineering, Hanyang University, South Korea.
1999–2001	Assistant Researcher, FINDTECH, South Korea. Internet Video Indexing (i-VIBS) Solution and Home Video Editor (i-CAM) Solution.
2001–2003	Software Engineer, MOBILEONE COMMUNICATIONS, South Korea. Net-Game Pack Solution and Mobile Game Development.
2003–2004	Software Engineer, MGAME, South Korea. Online Bomberman Game Development.
2005–2006	Researcher, ANY Corporation, South Korea. TFT LCD Monitor for G.E and Siemens Medical Solutions and Digital Video Recorder (DVR) Solution.
2006–2007	M.Eng, Electrical and Computer Engineering, Cornell University, USA.
2007–2008	Visiting Scientist, Computer Science, Cornell University, USA. Live Objects Project, sponsored by Prof. Ken Birman.
2008–2015	Doctoral Student, Computer Science, UCLA, USA.
2010–2010	Visiting Student Researcher,USC Information Sciences Institute, USA. Resource Allocation for Cloud Computing Project, sponsored by InfoSys.

2011–2012	Adjunct Staff, RAND Corporation, USA. Mobile Support for Healthcare Project.
2012–2013	Graduate Student Researcher,, UCLA Computer Science, USA. under Prof. Miodrag Potkonjak.
2013–present	Staff Software Engineer, Samsung Research America, USA. Cloud Research Lab.

PUBLICATIONS

Krzysztof Ostrowski, Ken Birman, Danny Dolev, and Jong Hoon Ahnn. *Programming with Live Distributed Objects.* 22nd European Conference on Object-Oriented Programming. 2008.

Jong Hoon Ahnn, Ken Birman, Krzysztof Ostrowski, and Robert. Van Renesse. *Using Live Distributed Objects in Office Automation.* ACM/IFIP/USENIX 12th International Middleware Conference. 2008.

Jong Hoon Ahnn, Uichin Lee, and Hyun Jin Moon. *GeoServ: A Distributed Urban Sensing Platform.* IEEE/ACM 10th International Symposium on Cluster, Cloud, and Grid Computing (best paper award). 2011.

Jong Hoon Ahnn and Miodrag Potkonjak. *What to Read? With Whom to Work? Where to Publish? - Scientific Techniques for Organizing and Conducting Engineering Research.* IEEE International Conference on Microelectronic Systems Education. 2011.

Sheng Wei, Jong Hoon Ahnn, and Miodrag Potkonjak. *Energy Attacks and Defense Techniques for Wireless Systems.* ACM 6th Conference on Security and Privacy in Wireless and Mobile Networks. 2013.

Jong Hoon Ahnn and Miodrag Potkonjak. *VeSense: Energy-Efficient Vehicular Sensing.* IEEE 77th Vehicular Technology Conference. 2013.

Jong Hoon Ahnn and Miodrag Potkonjak. *VeSense: High-Performance and Energy-Efficient Vehicular Sensing Platform.* Journal of Pervasive and Mobile Computing. 2013.

Jong Hoon Ahnn and Miodrag Potkonjak. *Modeling Mobile Cloud Computing Using Greenmetrics.* USENIX 8th Workshop on Feedback Computing. 2013.

Jong Hoon Ahnn and Miodrag Potkonjak. *Toward Energy-Efficient and Distributed Mobile Health Monitoring Using Parallel Offloading.* IEEE 35th Engineering in Medicine and Biology Society. 2013.

Jong Hoon Ahnn and Miodrag Potkonjak. *mHealthMon: Toward Energy-Efficient and Distributed Mobile Health Monitoring Using Parallel Offloading.* IEEE Journal of Medical Systems. 37(5). 2013.

Arun Jagatheesan, Jong Hoon Ahnn, Thomas Phan, Abhishek Singh and Juhan Lee. *Hierarchical Automatic Speech Recognition Powered by Data Infrastructure.* IEEE Consumer Communications and Networking Conference. 2014.

Jong Hoon Ahnn. *Toward Personalized and Scalable Voice-Enabled Service Powered by Big Data.* IEEE International Conference on Big Data. 2014.

Jong Hoon Ahnn. *Scalable Big Data Computing for the Personalization of Machine Learned Models and its Application to Automatic Speech Recognition Service.* IEEE Workshop on Scalable Machine Learning: Theory and Applications. 2014.

Jong Hoon Ahnn. *Big Data Computing for the Personalization of Services and its Application to Automatic Speech Recognition.* IEEE/ACM Symposium on Big Data Computing. 2014.

CHAPTER 1

Introduction

We have observed that rising popularity of smartphones with onboard sensors (e.g., GPS, compass, accelerometer) and always-on mobile Internet connections via WiFi or 3G/4G-LTE shed lights on using smartphones as a platform for large-scale mobile sensing and computing. According to the report [1], the number of phone users worldwide has been constantly growing: 4.08 billions(B) in 2012, 4.33B in 2013, and 4.55B in 2014. Only smartphone users in 2014 reached 1.75B as they become more affordable and the wireless network technology advances. The future estimation in the growing number is still positive and some optimist expects mobile devices including phones will surpass the number of population on earth very soon [2]. GeoServ provides a rough math for network bandwidth and storage requirements for 10 million mobile users [3]. For instance, 100 million mobile users could generate sensor data at the rate of 1KB/s per user (e.g., GPS, accelerometer, WiFi scanning data) and also send queries, requiring networking systems with a sheer amount of bandwidth (>800Gbps), storage space (>360TB/hr.), and corresponding computational power. The amount of computation required is difficult to estimate as it depends on characteristics of applications. Recently, Nokia researchers presented challenges for mobile data that become bigger than ever [83]. From Cornell researchers [84], the total amount of data on earth in 2010 exceeded one zettabyte (ZB). By end of 2011, the number grew up to 1.8 ZB. Further, it is expected that this number will reach 35 ZB in 2020.

In the area of sensors including mobile devices and other embedded sensors, BBResearch [86] reports the global market was around $56.3 billion in 2010 and grew to

$62.9 billion in 2011. The market is expected to increase to $91.5 billion by 2016 at a compound annual growth rate of 7.8%. Researchers in the area of urban sensing, vehicular sensing, and pervasive computing have studied fundamental challenges in mobile devices. There is a consensus upon needs for *scalable sensor networking systems* that can facilitate information sharing among billions of mobile users via always-on 3G/4G-LTE connections. With mobility come its inherence problems such as resource scarceness, finite energy, and low connectivity [4]. One promising design option would be using a mobile-to-mobile overlay network of 3G/4G-LTE users. Rybicki et al. [5, 6] proposed PeerTIS, where mobile phones on the road form a distributed hash table (DHT) to realize scalable information sharing in vehicular environments (e.g., congestion notification). However, mobile-to-mobile networking is not practical for several reasons. Most importantly, P2P connections between mobile devices are typically hampered by network address translation (NAT), a commonly used technique in the mobile operator's domain to better utilize limited IP address blocks and to provide secure Internet connectivity [7]; for P2P we need additional services such as session initiation protocol (SIP) or P2P proxy servers [8]. Moreover, P2P protocol operations such as routing and searching may require quite a few message exchanges over mobile nodes, which results in intolerable delays given that cellular networks typically has a large round trip delay (i.e., several hundred milliseconds [9]). Also this causes significant resource consumption (e.g., battery, processing power, and bandwidth), which is in turn a serious problem for resource-limited smartphones. For this reason, recent work puts more focus on Internet server-based urban sensing schemes such as GeoServ [3], VeSense [10] [11], mHealthMon [12] [13]. The distributed file storage however works well only for typical urban sensing applications that perform lightweight data operations. The DHT-based data-fetching scheme is too slow when a mobile application requires the large number of files to read from distributed servers for processing. It is obvious that computationally intensive applications are hard to tolerate such delays from data read/write operations under unstable wireless connections. The next sec-

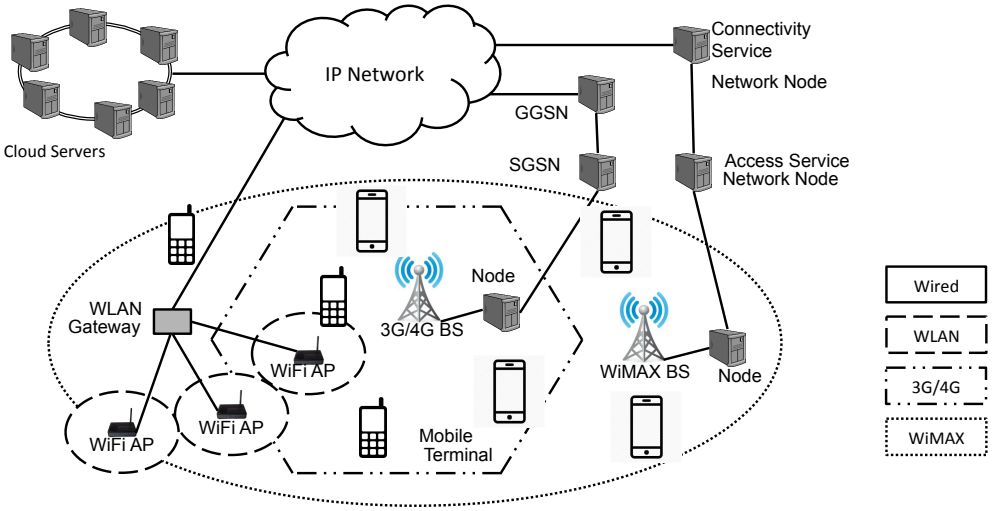

Figure 1.1: An example of mobile cloud computing, where mobile phones can connect to cloud servers via various network interfaces such as WiFi, cellular network, and WiMax.

tion continues the discussion on limitations on existing schemes and introduces new opportunities in mobile sensing and computing.

1.1 Related Work

The paradigm of mobile computing is different from the one of urban sensing, vehicular sensing, and pervasive computing in such a way that more computationally expensive applications such as video games, speech synthesis, natural language processing, augmented reality, and wearable computing are into consideration in mobile computing. Consider a mobile cloud computing scenario presented in Figure 1.1. We observe that considerable efforts can be found in solutions to address the challenges of computational power and battery lifetime by offloading computing tasks to cloud. In the last decade, many researchers have proposed several approaches: Client-server(RPC) offloading (Spectra [20], Chrome [21]), peer-to-peer offloading (Hyrax [22]), code offloading (Scavenger [19]), VM migration (MAUI [14], Cloudlets [15], CloneCloud [16], MobiCloud [17]), and more recently application server based parallel offloading (ThinkAir [18]).

Provided is a comprehensive survey from the viewpoint of data-intensive computation in ten major MCC systems. Typical MCC surveys often constitute network protocol, code rewrite requirement, offloading granularity, profiling, resource monitoring, and software preinstallation requirement. Our survey differs from typical MCC surveys mainly in several ways: scalable big data and computing support, fault-tolerant offloading service. The extended properties in our survey include parallel offloading, multi-cloud support, disconnected operation, reliable message delivery, cloud sever scalability, and cloud storage access. Table 1.1 presents the comparisons of existing systems through 15 properties as we consider them important factors to build the MCC system. The most recent works exploit VM migration offloading to the cloud. However, it is costly in initiating a VM instance upon an offloading request from a mobile device. *Our proposed system called BigMobile exploits simplicity from the client-server approach, but relaxes software pre-installation constraint for offloaded tasks.* An offloaded task with associated data and parameters are transmitted to the cloud in the same manner that the VM migration does, but no VM instantiation happens. We develop a computation offloading service controller in the cloud and it takes charge of offloading task's lifecycle from the beginning to the end as detailed in Section 2.4.

Most of aforementioned systems offload one of the computational blocks (methods, classes, threads, bundles, or objects) to the cloud in a sequential manner. This means that only one offloading can be made at a time. Once the offloading originator has a result back from the cloud, it can initiate another request afterward. It is obvious that the sequential offloading is inefficient when concurrent multiple requests can be made by the dependency analysis of computational blocks. There are few researchers who tackle the inefficient offloading problem by parallel offloading in various contexts. Ahnn et al. proposes VeSense [10] [11] that exploits concurrent offloading method in vehicular sensing applications while mHealthMon [12] [13] exploits the similar approach that can save energy for a mobile health monitoring application. *BigMobile extends the parallel offloading approach implemented in VeSense and mHealthMon to more scal-*

able bit data access and computing support. ThinkAir [18], one of the pioneers in this direction, proposes application server-based parallel offloading implemented with Android NDK [35], different from the VM-based approach in MAUI [14], Cloudlets [15], CloneCloud [16], MobiCloud [17].

Spectra [20], Chrome [21], and ThinkAir [18] require code rewriting as well as offloading method/thread/bundle/object preinstallation to smoothly support computation offloading. *BigMobile requires code rewriting through the proposed programming interface, however it does not require the software preinstallation in cloud for offloading.*

For the cost model, there are three factors to consider: latency, energy, and budget. Spectra [20], Chrome [21], Hyrax [22] Scavenger [19] only consider latency in their cost modeling. MAUI [14], Cloudlets [15], and CloneCloud [16] consider energy and latency at the same time. *The cost model of BigMobile is similar to the one of CloneCloud [16], where an objective function in the solver optimizes tradeoffs between mobile execution and cloud execution w.r.t. energy and execution time while meeting energy and latency constraints*. Unlikely, MobiCloud [17] takes both energy and budget into account in its cost model.

For the granularity of computation offloading, there are three categories to classify schemes. The first category is code-level offloading which is exploited by MobiCloud [17] and Scavenger [19]. The second category is method-level offloading which is exploited by Spectra [20], Chrome [21], MAUI [14], and ThinkAir [18]. *BigMobile falls in this category.* The third category is bundle/object/module-level offloading which is exploited by Cloudlets [15] and Hyrax [22]. The last category is thread-level offloading in which CloneCloud [16] falls onto this bucket.

For profiling, there are two methods: history-based and stochastic profiling. Most of recently published work (Scavenger [19], Spectra [20], Chrome [21], MAUI [14], ThinkAir [18], Cloudlets [15], CloneCloud [16]) relies on the former method. *BigMobile's profiler is similar to them*. MobiCloud [17] exploits a stochastic profiling method.

		Network protocol	Code rewrite required	Disconnected operation	Parallel offloading	Multi-clouds support	Cloud storage access	Reliable msg delivery	Offloading granularity	Profiling	Resource monitoring	Server scalability	Cost model parameter	software preinstallation
Client-server	Spectra [20]	WiFi	O	X	X	X	X	X	method	history	X	X	latency	O
	Chrome [21]	WiFi	O	X	X	X	X	X	method	history	X	X	latency	O
	ThinkAir [18]	3,4G WiFi	O	X	O	X	X	X	method	history	O	O	energy, latency	O
	BigMobile	3,4G WiFi	O	O	O	O	O	O	method	history	O	O	energy, latency	X
VM migration	Cloudlets [15]	WiFi	X	X	X	X	X	X	bundle	history	O	O	energy, latency	X
	MAUI [14]	3,4G WiFi	X	X	X	X	X	X	method	history	O	O	energy, latency	X
	CloneCloud [16]	3,4G WiFi	X	X	X	X	X	X	thread	history	X	O	energy, latency	X
	MobiCloud [17]	WiFi	X	X	X	X	X	X	code	stochastic	X	O	energy, budget	X
Peer-to-peer	Hyrax [22]	WiFi	X	X	X	X	X	X	module	N.A.	O	X	latency	X
	Scavenger [19]	WiFi	X	O	X	X	X	X	code	history	O	X	latency	N.A.

Table 1.1: Comparisons of major computation offloading systems.

1.2 Toward Data-Intensive Computation Offloading

Shirax et al. categorizes major research issues in mobile cloud computing in two folds: augmented computing power and cloud contents [23]. The former has been extensively studied while researchers have put little attention on the latter. Often, researchers make assumptions that there exist cloud storage services such as Amazon S3, Dropbox, Google cloud storage, and many others so that scalable file system can be easily accessible on demand. On the other hand, we observe that mobile application developers and organizations prefer to own their data storage from the early stage of development. For instance, most of Silicon Valley startups either plan to develop or already have their own data storage and analytics platform as data become a critical asset in building intelligence along with mobile applications. The trends say that easy accessibility to cloud contents in MCC emphasized by [23] becomes more important than ever from a practical point of view. We believe that the issue can open a door to a new era of mobile cloud computing toward big data. Associated research questions are following: how offloaded computation can benefit from big data stored in distributed file systems or database; how to fuse augmented mobile cloud systems with existing ever-successful big data platform such as Hadoop.

Some big companies' use cases can be easily found in these days. Yahoo has more than 100,000 CPUs in over 40,000 servers running Hadoop, with its biggest Hadoop cluster running 4,500 nodes, and 455 petabytes (PB) of data warehouse [31]. Facebook reports its 300 PB based data warehouse [28]. Google claims that they scan more than 20 PB of data per day [30]. Netflix recently reports they have 10 PB data warehouse running on top of Facebook's Presto open source software [27]. Boeing reports a jet engine generates 10 terabytes (TB) of data every 30 minutes [85]. A single six-hour flight would generate 240 TB of data. Mobile applications built on top of data analytics are naturally to require the large amount of data. A good representative class of applications would be machine learning-driven mobile applications that require both the consider-

able amount of computation and data at the same time. Examples are face recognition, speech recognition, handwriting recognition, data analytics, natural language processing, video processing, multimedia search, augmented reality [32] [33] [34].

1.3 Contribution

To this end, we propose a data-intensive MCC platform called BigMobile.

- First, a survey provides 15 viewpoints of properties of data-intensive MCC by considering ten major existing MCC systems. Our survey differs from typical MCC surveys in several ways: scalable big data and computing support and fault-tolerance of MCC systems.
- Second, the proposed system exploits an function-level parallel offloading method in a client-server paradigm with no requirement of software preinstallation for offloading tasks.
- Third, BigMobile supports a set of interfaces for the fuse of MCC with ever-successful big data platform Hadoop. The big data integration provides two benefits: integration of job scheduler and native access to distributed file system and possibly to no SQL database. To our knowledge, no existing MCC system seamlessly supports big data platform.
- Fourth, offloading message delivery is achieved via our message queue design, which aims being scalable, reliable, but fast in a distributed manner. It provides the functionality of a messaging system for offloading requests. Exploiting asynchronous communication between mobile and cloud can achieve fault-tolerance under wireless network instability.
- Fifth, a BigMobile solver uses data collected by a profiler as input to a global optimization problem that determines which remoteable methods should execute

locally or remotely among multiple clouds choices. The solver's goal is to optimize the tradeoffs between mobile execution and cloud execution w.r.t. energy and execution time for a given mobile application and multiple clouds.

- Sixth, we provide a couple of evaluation results in terms of time saving and energy saving based on two types of benchmarks: computationally-intensive applications and data-intensive applications. For the former, we consider chess game, puzzle game, cryptographic algorithm, and Huffman compression algorithm, for the latter, we consider the state-of-the-art speech recognition application. The evaluation extends a single offloading to parallel offloading. The further evolution of data-intensive computational cloud is provides in several aspects: performance comparison of Hadoop version 1 and version 2, Hadoop overhead, benefit of file compression, and performance of cloud storage to support small/large files. Finally, the performance of the proposed message queue is presented.

1.4 Thesis Organization

This thesis is organized as follows; Chapter 2 presents common assumptions of mobile cloud computing, the design goal of the proposed system BigMobile, its system overview, and the flow of data-intensive computation offloading; Chapter 3 presents programming abstraction for BigMobile applications including interface definition, interface implementation, exposing methods to mobile clients, and method invocation; Chapter 4 introduces the problem of parallel offloading to multiple clouds, partitioning algorithm, and profiling techniques; Chapter 5 introduces one of the representative data-intensive mobile applications: automatic speech recognition (ASR) and its computation for personalization; Chapter 6 presents the overall system architecture for data-intensive cloud support including reliable message queue, RESTful data service, and RESTful computation service; Chapter 7 presents two types of evaluation of BigMobile system: simple benchmark mobile applications and data-intensive mobile application.

The chapter extends the performance evaluation in two aspects: energy saving and time saving for both mobile and clouds; Chapter 8 concludes this thesis by summarizing contributions, and Chapter 9 lists tentative future work regarding Tizen OS port, security and privacy, and context awareness.

CHAPTER 2

BigMobile System Architecture

This chapter is organized as follows: common assumptions of mobile cloud computing, the design goal of the proposed system BigMobile, its system overview, and lastly the flow of data-intensive computation offloading.

2.1 Common Assumptions in Mobile Cloud Computing

Common assumptions in existing MCC systems are surveyed and presented through Table 1.1. These are following,

First, always-on and reliable wireless connection with no consideration of disconnection in the middle of offloading processing [25]. We revisit this as wireless networks are flaky and versatile for mobile devices on the move since we believe that MCC systems require a fault-tolerant network communication. We tackle this problem with an asynchronous nature of connection.

Second, the performance of computing in cloud VMs is assumed to be all same when scheduling and scaling, assuming that VM instantiation and applications running on VMs are fast enough than client-server or VM-based offloading scheme as shown in Section 1.1. We revisit this assumption as we consider multiple clouds that may have various hardware resource configurations and in turn result in various performance deviation. We tackle this problem by providing a light-weight cloud provision idea with no VM and no software preinstallation for offloading tasks in cloud.

Third, an implicit cloud contents access mechanism is often assumed in almost all

existing MCC proposals as found in Section 1.1. The availability of cloud contents is very different from accessibility of them in reality. The assumption of cloud contents that are readily available from the cloud must be revisited due to the large volume of data and retrieval time [23] in data-intensive MCC applications. We tackle this problem by providing an explicit cloud storage interface to distributed file systems in cloud.

2.2 BigMobile Design Goals

The design goals of BigMobile are based on design principles from the aforementioned literature and new observations, which will help MCC systems overcome major challenges to build more reliable and scalable MCC system. We pursue a common set of MCC design principles: dynamic adaptation to changing environments, ease of use for developers; performance improvement including energy and delay through exploiting cloud computing resources; and dynamic scaling of computational power [26]. We reflect these observations in BigMobile through the following key design principles that in turn become core assets of our work.

Fault-tolerant communication, where mobile devices connect to clouds via an asynchronous connection to tolerate flaky network conditions. The established connection between a mobile device and cloud is disconnected right after an offloading request is successfully sent to the cloud. When the cloud finishes the offloaded task, a notification service running in the mobile device is capable of capturing a job competition notification. In this manner, BigMobile can tolerate the problem of wireless network instability. The asynchronous property can nicely relax the hard assumption of always-on connection that is commonly assumed by existing MCC systems as detailed in this chapter. For MCC application developers, this is achieved by our proposed programming abstraction.

Programming abstraction, where mobile application developers can easily build MCC applications by hiding lots of underlying complexities regarding how to partition

applications, how to send offloading message to cloud, how to assign cloud resources to the offloaded tasks, how to access cloud storage upon the offloading request, many other system issues including reliability and scalability as detailed in Chapter 3.

Application partitioning problem with multi-clouds, where BigMobile provides a flexible solver for mobile devices to reduce both energy and time by computation offloading. Compared to existing MCC systems, BigMobile provides the first system-level implementation of parallel offloading to multi-clouds. BigMobile's solver formulates the multi-clouds partition problem as the 0-1 Integer Programming (IP) Problem as detailed in Chapter 4.

Server-side fault-tolerance and scalable offloading message delivery, where we propose and implement a reliable and fault-tolerant offloading message queue. We adopt a traditional approach of message queue in a consumer/producer paradigm, and extend it to be reliable by having multiple brokers in the middle to keep multiple copies of incoming offloading requests in them. The design of multiple brokers has several benefits: it can tolerate a broker failure; it can horizontally scale out as the numbers of offloading requests grow by simply adding more brokers as detailed in Section 6.1.

Big data platform integration, where BigMobile provides a way to fuse with ever-successful big data platform called Hadoop. The integration enables to utilize a resource scheduler (YARM) in Hadoop [71] upon the mobile job submission thru a proposed offloading interface. In addition, we enable and implement an interface to access to HDFS (Hadoop distributed file system) so that ever-growing demands in mobile applications with computationally expensive features such as machine learning can be energy-effectively supported as detailed in Chapter 6. We evaluate BigMobile with a state-of-the-art automatic speech recognition application, which requires the considerable amount of both computation and data in Chapter 7.

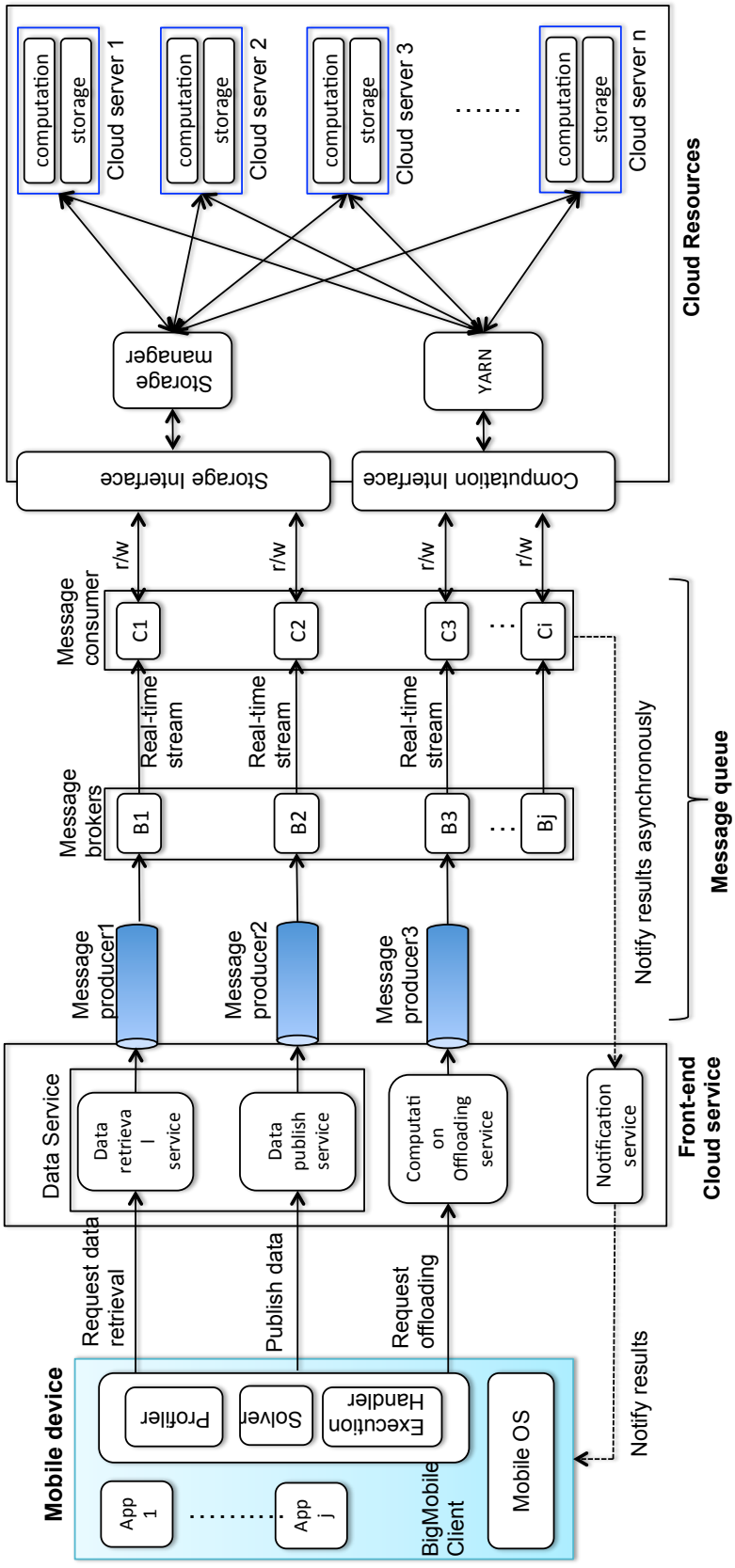

Figure 2.1: A high-level overview of BigMobile system architecture is presented.

2.3 BigMobile System Overview

Figure 2.1 provides a comprehensive and high-level overview of the proposed BigMobile architecture. On the mobile device, the BigMobile client consists of three core components.

Profiler. it instruments a mobile application and harvests energy and time of each computation task and network profile as detailed in Section 4.4.

Solver. an optimization problem solver as an offloading decision engine to optimize tradeoffs between mobile execution and cloud execution w.r.t. energy and execution time based on a weighted objective function. The real optimization algorithm runs on a mobile device; however it can be offloaded to the offloading service in the cloud as an alternative to save energy consumption as typical computation offloads as detailed in Section 4.2.

Execution handler. the handler controls two services: data service sent to the cloud for uploading/fetching data to/from the cloud; offloading service to offload computation to the cloud as detailed in this chapter.

The aforementioned three component in mobile devices closely incorporates with the BigMobile cloud. On the cloud, BigMobile constitutes six core components.

Data service. a data (publish/retrieval) service handles mobile user's requests for data query, and roles as a interface to cloud storage. This incorporates with the execution handler sitting on the mobile device. This is achieved by programming abstraction detailed in Chapter 3.

Computation offloading service. a mobile computation offloading service controls an entire flow of offloading based on an optimal execution solution computed from the solver to save both energy and time as detailed in Chapter 4 , Section 6.4, and Chapter 6.3.

Message queue. a producer/consumer design for reliable message delivery mech-

anism guarantees fault-tolerant offloading request/response communication between mobile devices and clouds, even when serious network failure occurs as detailed in Section 6.1.

Cloud storage interface. Fourth, a uniform and easy interface enables for a mobile device to seamless access to the cloud storage via RESTful APIs via one of the widely used Internet protocol HTTP(s). The RESTful APIs supports a simple set of commands *PUT*, *GET*, *DELETE* for easy data access operations as detailed in Section 6.2.

Cloud computation interface. a native APIs interface to cloud computation resources, wherein a mobile application can easily submit a set of parallel computation offloading requests to the cloud as detailed in Section 6.3.

Security. For security, the BigMobile clients leverage Kerberos to perform user authentication on all remote computation execution. Kerberos is a open source network authentication protocol and is designed to provide strong authentication for client/server applications by using secret key cryptography and deployed in a number of large scale systems [29].

The way that BigMobile works is complex. Therefore, we describe The overall workflow of the offloading procedures in BigMobile in Section 2.4.

2.4 Flow of Data-Intensive Computation Offloading

A sequence diagram for dynamic application offloading in the BigMobile system architecture is depicted in Figure 2.2, where 15 steps of interactions between a mobile device and data-intensive computational cloud are described in order. (1) A *profiler* is activated with a mobile application, and then it preforms a comprehensive profiling including (2) network and (3) software/hardware, followed by (4) the activation of an *optimization solver*, which is in charge of decision making for whether given computation is offloaded or not. (5) An *execution handler* controls a whole flow of computation in a mobile device including sending out offloading requests to the cloud. It also re-

ceives notifications for the offloading requests from the cloud regardless that the request is successfully processed or failed. (6) A combination of the asynchronous offloading mechanism and *notification service* enables BigMobile to tolerate unstable wireless network connection. For instance, suppose the wireless connection is disabled right after submitting an offloading request successfully to the cloud. A mobile device will receive a notification message with the error or success code as soon as the connection is recovered. (7) BigMobile proposes and implements a reliable offloading *message queue* consisting of consumers, brokers, and producers. The producer forwards incoming requests (or messages) to one of the brokers in a sequential manner, where the broker replicates messages to a certain number of other brokers. The replication factor is configurable, and typically we set it to 3, meaning that three copies of messages are always kept in brokers. Even when one broker fails, two remaining brokers maintain original messages. A set of consumers provides a way to real-time streaming for offloading request or data service requests. The message queue aims at scalability and flexibility: It allows scaling out in cloud-back end as the number of requests grow; it is flexible in hosting various applications' needs in terms of real time request processing by adding more consumers with different functionalities such as database access, and real-time analytics. As a result, BigMobile guarantees reliable message delivery and processing in a scalable manner. (8) The *computation offloading service* in the cloud provides a response to the original request, (9-12) and then the *execution handler* in mobile devices is allows to upload associated data with the offloading computation to the *cloud storage*, (13) where uploaded data will be fetched upon execution in the cloud. The *computation interface* will request a resource allocation to Hadoop's resource manager called YARN, where it consists two main components: a scheduler and application manager; The scheduler is responsible for allocating resources to the offloading tasks subject to familiar constraints of capacities, queues; The application manager is responsible for accepting job-submissions, negotiating the first container for executing the application specific application master and provides the service for restarting the application master

container on failure. Hadoop details are found in [71]. (14) Upon the offloaded task completion, (15) the *notification service* in the cloud front-end will provide a task competition notification back to the request originator, and this achieves the data-intensive computation offloading in an asynchronous and reliable manner. This completes the end to end offloading from mobile to cloud in BigMobile.

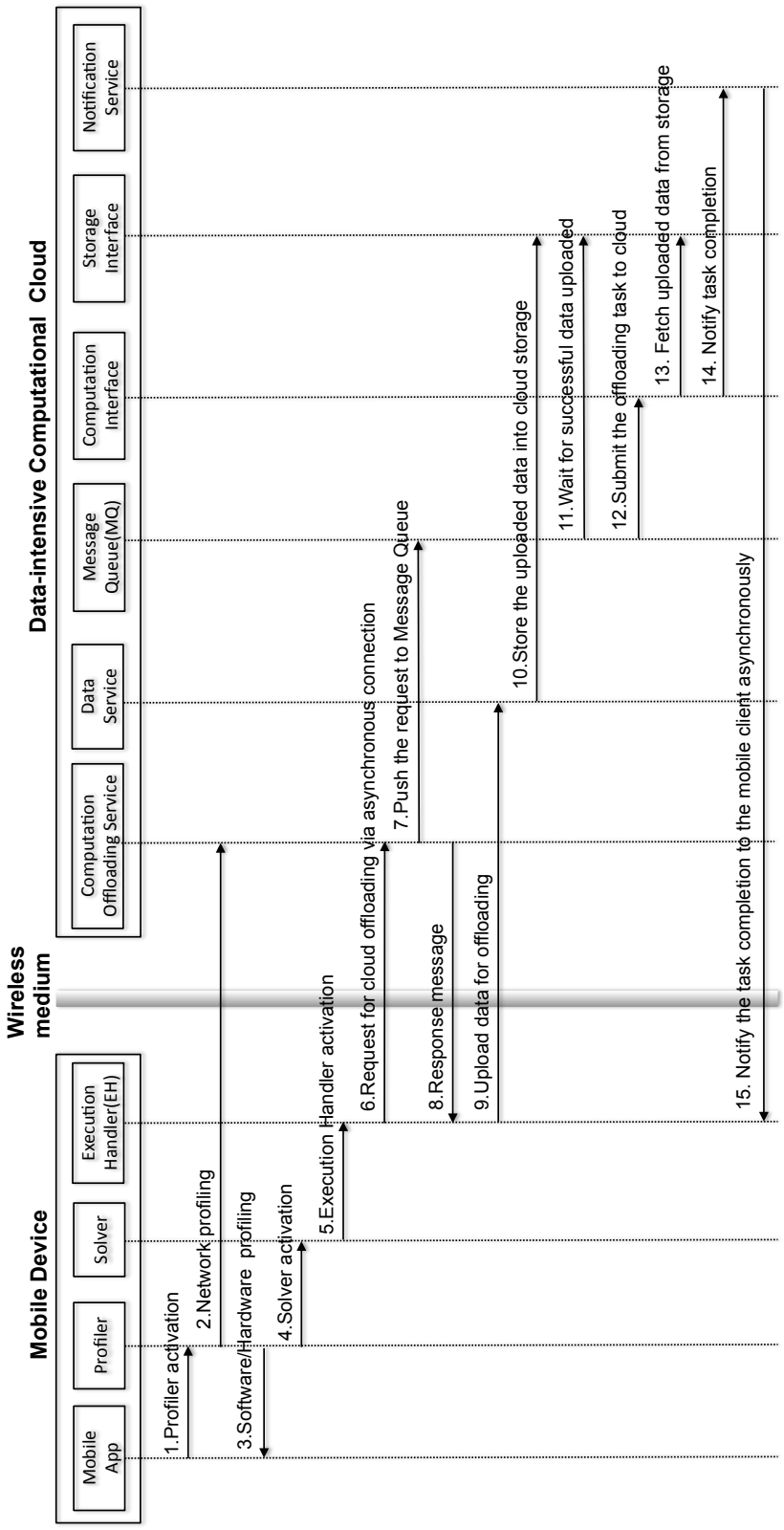

Figure 2.2: A sequence diagram for asynchronous application offloading in BigMobile.

CHAPTER 3

Programming Mobile Cloud Application

This chapter is organized as follows: programming abstraction for BigMobile applications including interface definition, implementation, exposing methods to mobile clients, and finally method invocation.

3.1 Programming Abstraction

We describe the detailed process by which an application developer writes code to make use of BigMobile. In order to facilitate mobile application developers in their production cycle, new tools such as compiler, programming interface are required so that they can support programming abstraction for complex MCC applications. The programming abstraction plays a key role to hide underlying complexity between mobile devices and data-intensive computation clouds. We aim to provide a practical, reproducible, but efficient programming interface for developers to control the behavior and execution location of the applications. Unlike many existing work mentioned in Section 1.1, ThinkAir [18] employs a promising and practical approach to provide a programming interface using AIDL (Android Interface Definition Language) [35] although very little information is disclosed. BigMobile adopts the similar approach to ThinkAir, but extends it to following ways:

- First, new implementation of asynchronous binding between mobile and cloud, since current implementation of AIDL only supports synchronous binding between mobile and cloud.

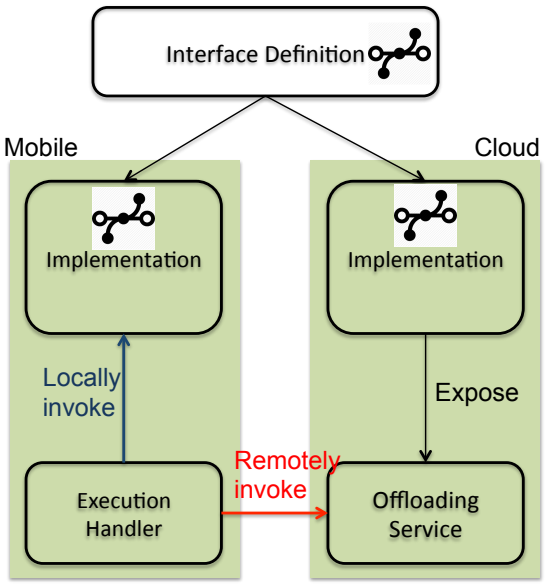

Figure 3.1: In BigMobile's programming abstraction, there are steps for developers to follow: interface definition, interface implementation, method exposure, and method invocation.

- Second, BigMobile relaxes the software preinstallation requirement in ThinkAir. BigMobile does not require the preinstallation for offloaded software. Rather, our system sends out required binaries for offloaded computation to the cloud on demand.

- Third, ThinkAir assumes that static application server in the cloud is always ready to accepting offloading requests from the mobile. Rather, the data-intensive computational cloud in BigMobile instantiates a resource slot and runs on a new offloading service on it so that the newly launched service can accept mobile's requests on demand.

- Fourth, the integration with big data platforms allows seamless access to cloud computation as well as to cloud storage on demand. We believe that our data-intensive computational cloud approach can provide a promising solution for the scalable and high-performance MCC back-end.

BigMobile exploits the inter-process communication mechanism, which is interface-based, similar to RPC but extends to lightweight and more importantly asynchronous. It uses a proxy class to pass values between the client and the remote service in cloud. As shown in Figure 3.1, there are two steps for developers to follow:

First step: implementing desired interface methods. The BigMobile compiler (a modified compiler on top of AIDL compiler) creates an interface in the Java programming language from the user-defined interface. This interface has an inner abstract class named *Stub* that inherits the interface (and implements a few additional methods necessary for the remote call). A programmer must create a class that extends *MyInterface.Stub* and implements the methods declared in the interface;

Second step: exposing the user-defined interface to clients from the remote service that is supposed to run on the cloud. If a programmer writes a remote service in the cloud, one should extend *Service* and override *Service.onBind(Intent)* to return an instance of her class that implements her interface.

3.2 Programmable Interface Definition

BigMobile application developers should create an interface first to make methods offloadable/remoteable to the cloud. An example for the interface *IOffloadingService* (line 2) is presented in Listing 3.1, where two methods *foo* (line 3) and *bar* (line 4) are defined. The methods can take zero or more parameters, and return a value or void. All non-primitive parameters require a directional tag *in, out, or inout* indicating which way the data goes. The app developers should be careful as marshaling parameters are expensive. Note that static fields will not be supported as an offloadable method in the interface.

Listing 3.1: An example of the interface definition for offloadable methods

```
1 // Declare the interface saved in IOffloadingService.aidl
2 public interface IOffloadingService {
```

```
3     int foo(in int param1, out String[] param2);
4     boolean bar(in long param1, out String param2, out List<int>
          param3);
5 }
```

3.3 Interface Implementation

Along the line of Listing 3.1, the Android NDK [35] provides a tool to generate IOffloadingService.java interface based on IOffloadingService.aidl. The interface *IOffloadingService* (line 1) generated in Offloading.java comes with a subclass *Stub* that is further defined an abstract implementation of its parent interface such as *IOffloadingService.Stub* that includes a few helper methods, *asInterface()*, which takes an *IBinder*. It returns an instance of the *Stub* interface. In order to implement the user-defined *IOffloadingService* interface, one must extend *Binder* interface and implement the remoteable methods inherited from the interface as shown in Listing 3.2. The first method *foo* (line 2) takes two parameters: one for input and the other for output, and returns integer value. The implementation of *bar* (line 6) takes one input and two output parameters, and returns Boolean value.

Listing 3.2: An example of the *IOffloadingService* interface (declared in Listing 3.1) implementation for remoteable methods

```
1 private final IOffloadingService.Stub myBinder = new
      IOffloadingService.Stub() {
2     int foo(in int param1, out String[] param2) {
3         // ... (code omited)
4         return 0;
5     }
6     boolean bar(in long param1, out String param2, out List<int>
          param3){
7         // ... (code omited)
```

```
8           return true;
9       }
10  }
```

3.4 Exposing interface to mobile clients

The implemented methods in the interface (Listing 3.3) can be exposed as a (remote) service by allowing a mobile application to bind to it. In order to expose the interface, a cloud server must extend *Service* (line 1) and implement *onBind()* (line 5) to return an instance of the given class that implements the generated *Stub* (line 7) as explained in Section 3.3.

Listing 3.3: An example of the Service implementation for remoteable methods

```
1  public class OffloadingService extends Service {
2      @Override
3      public void onCreate(){ super.onCreate(); }
4      @Override
5      public IBinder onBind(Intent intent) { return myBinder;
6      }
7      private final IRemoteService.Stub mBinder = new IRemoteService
           .Stub() {
8          // Refer to Listing 3.2 for details of implementation
9      }
10 }
```

3.5 Offloadable Method Invocation

The current implementation of inter-process calls in Android NDK [35] is based on synchronous communication between mobile and server. However, BigMobile modifies the binder interface IBinder to make it *asynchronous*. The modified implementation

relaxes a hard requirement for maintaining a connection while waiting for a response to an offloading request. Right after sending out an offloading request successfully from mobile to cloud, the connection is unbinded. The notification service in the cloud front-end will inform resultant messages from the cloud to the mobile once the offloaded execution is complete.

As shown in Listing 3.4, there are a few steps a calling class should make to call the remote interface:

- First, declare a variable of the interface type that the .aidl file defined.
- Second, implement *ServiceConnection* (line 7).
- Third, call *Context.bindService()*, passing in the *ServiceConnection* implementation (line 7).
- Fourth, in the implementation of *ServiceConnection.onServiceConnected()*, one will receive an *IBinder* instance (line 8).
- Fifth, call *IOffloadingService.Stub.asInterface ((IBinder)service)* to cast the returned parameter to the interface type (line 9).
- Sixth, call the methods that a user defined on the interface.
- Seventh, one should trap *DeadObjectException* exceptions, which are thrown when the connection has broken; this will be the only exception thrown by remote methods.
- Eighth, unlike the original Android NDK's implementation, BigMobile automatically calls *Context.unbindService()* to disconnect from cloud after the offloading method call is made.
- Ninth, once the offloading request is processed from the cloud, the callback function *IOffloadingServiceCallback* will be called as a mean of notification of task

competition. The callback with the return value achieves the asynchronous offloading communication between the mobile and the cloud. Thereby, the application allows to proceed further tasks based on the result.

Next chapter, we continue to explain how we formulate a partitioning problem and how the MCC application is partitioned and offloaded to multiple clouds.

Listing 3.4: An example of the BigMobile application and offloading method invocation after binding to cloud

```
1  public staic class BigMobileApp extends Service {
2      IOffloadingService service =null;
3      int x=0; String[] list; long y= 1; String str=      ; List<int
           > list2=new List<int>();
4      @Override
5      protected void OnCreate(Bundle savedInstanceState) { // (
           omitted)
6          bindService(new Intent(IOffloadingService.class.getName(),
               conn, Context.BIND_AUTO_CREATE);
7                  }
8                  private ServiceConnection conn = new
                       ServiceConnection(){
9              public void onServiceConnected(ComponentName className
                   , IBiner svc){
10                 service = IOffloadingService.Stub.asInterface(svc)
                       ;
11                 try{
12                     service.registerCallback(callback);
13                     service.foo(in x, out list);  service.bar(in x
                           , out str, out list2);
14                 } catch(RemoteException e){}
15             } public void onServiceDisconnected(ComponentName
                   className){
16                 service=null;
17             }
```

```
18          };
19          private IOffloadingServiceCallBack callback = new
                IOffloadingServiceCallBack.Stub(){  //(..omited..)
20          };
21  }
```

CHAPTER 4

Partitioning Mobile Application

Traditionally, computation offloading is assumed to plan a remote task on a single remote server, also known as a remote procedure call (RPC). Seemingly, the client-server approach such as MobiCloud [17] and Scavenger [19] falls onto this case as extensively discussed in Section 1.1. The VM-based offloading approach (e.g., MAUI [14], Cloudlets [15]) can be more flexible in dealing with multiple resources in the cloud, but it makes an underlying assumption that all the VMs may have very similar properties in terms of the performance of computation for a given task. The translation of the assumption seems that VMs may run on machines that have similar hardware configurations such as CPU, memory, storage, and other VM settings, so thus on a single cloud. This could be misleading to formulate a program partitioning problem when considering multiple cloud options, which are common in these days.

In modern data-intensive computing platform such as Hadoop, data is distributed across multiple clusters, where a task scheduler (or resource manager) for each cluster disjointly locates tasks on its exclusively configured cluster, and distributed file systems are often limited by the boundary of each cluster for typical tasks. In companies, it is typical to several clusters from staging clusters to production clusters, even from one production cluster to the other production cluster. For instance, it is not realistic to accommodate 40,000 servers in one single cluster [31]. This trends stay similar to other companies such as Facebook [28], Google [30], and Netflix [27]. We thus observe that the distributed placement of both data and computation across multiple clouds becomes common practice in these days for data-intensive computing. This

behooves us to formulate multi-clouds partitioning problem, where computation in a mobile application is partitioned into multiple splits, for them to run on multiple cloud resources. In BigMobile, the granularity of clouds is based on the number of cloud task schedulers in clouds, where each task scheduler such as YARN in Hadoop is in charge of task (or offloaded computation) scheduling. For example, an organization has two Hadoop cluster with different configurations, and the mobile application is supposed to be partitioned into 1+2 chunks so that the first partition is for mobile execution and the other partitions are assigned to two Hadoop clusters, respectively.

This chapter is organized as follows: the introduction to offloading to multiple clouds, the problem formulation of parallel offloading to multiple clouds, partitioning algorithm, and profiling techniques at the end.

4.1 Offloading to Multiple Clouds

Programing partitioning to multiple clouds is controlled by an *execution handler* along with a profiler and optimization problem solver in a mobile device as detailed in Section 2.4. Parallel offloading to multiple clouds scenario is presented in Figure 4.1: method a sent to cloud 1 or sent to cloud 2. We assume that each cloud may or may not have distinct source of data as denoted by data1 and data2 being typical for data-intensive computing in these days.

We formulate the aforementioned multi-clouds partitioning problem in a directed graph, where a set of nodes in the graph represents computation, specifically a method in a mobile application, and edges represent the interaction such as method invocation between nodes. We specify a value of weight for an edge indicating communication cost in the unit of power consumption, translating into the amount of data to transfer from one vertex to another. The goal of the *optimization solver* in a mobile device is to optimize the tradeoffs between mobile execution and cloud execution w.r.t. energy and execution time by computation offloading, while meeting two constraints such as

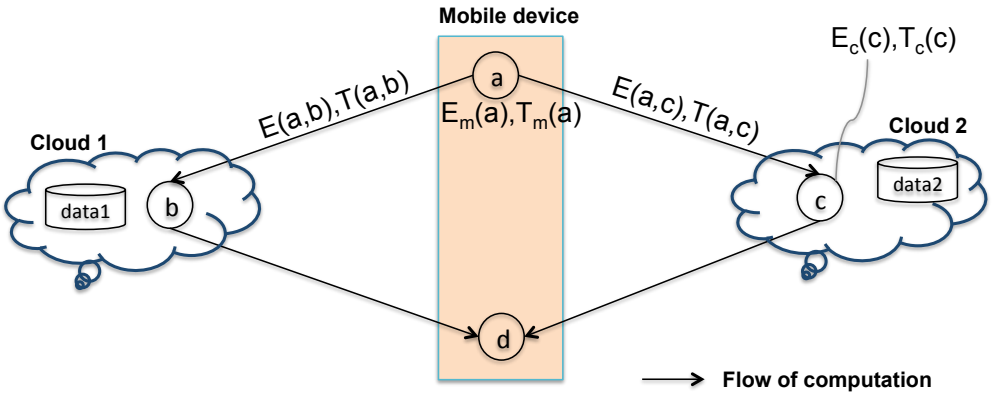

Figure 4.1: Presented is a parallel and multi-clouds offloading scenario for a mobile application in BigMobile. a, b, c, and d represent methods for a mobile application.

the total execution time limit and energy consumption limit. The partition problem can be solved by calculating the costs of the partitioning nodes in the graph with two major factors: the cost of computation (calculated from weights of nodes) communication (calculated from weights of edges). For the given method a, computation cost on cloud 1 is different from that on cloud 2 as we consider multiple cloud settings. The optimal partitioning result solved as an 0-1 integer programming provides an optimal choice of methods to offload to one of the multiple clouds.

4.2 Problem Formulation

We present the graph construction, problem formulation, and profiling techniques in this section.

4.2.1 Graph Construction

More formally, we construct a graph $G = (V, E)$, where each vertex $v \in V, u \in V$ represents a method or function in the call graph, and an edge $e = (u, v) \in E$ represents an invocation of method v from method u . A set of partitions denoted by $P = \{p_0, p_1, ..., p_n\}$ represents the n multiple clouds to host offloading requests. As a

part of static analysis of the mobile application, we utilize Soot [36] to characterize the function call graph G, and it generates a static profile of the application. The *profiler* collects computation profiles of the application by one time analysis to set the values of vertices, and monitors network profiles at runtime to set the values of edges. In Figure 4.1, $E_{u \to v}$ on the edge $e = (u, v)$ represents the amount of energy required for sending data from vertex u to vertex v over wireless network which is typical in mobile cloud computing environments. Similarly, $E(v)$ represents the amount of energy needed for computing method v on either mobile or cloud. We differentiate one notation of mobile execution $E_m(v)$ from the one of cloud execution $E_c(v)$. The amount of execution time required for computation and network data transmission is denoted by $T_{u \to v}$, $T(v)$ and more detailed notations by $T_m(v), T_c(v)$ in the similar manner as we define the notation of the energy consumption.

4.2.2 Problem Formulation

In modeling computation partitioning to multiple clouds, decisions must be made in terms of tradeoffs between mobile execution and cloud execution w.r.t. energy and execution time. As an example, a mobile application must decide which of methods in G to offload for satisfying the weighed optimized objective between time and energy to one of the cloud resources. The decisions to be made are which method is to be offloaded, and which cloud service is selected to serve offloading requests. Those two questions are reflected into the objective function we will define later in this section.

The objective is to optimize the tradeoffs between mobile and cloud execution for a given BigMobile application. For simplicity, we make following assumptions which will be relaxed later as we design reliable and fault-tolerant end-to-end BigMobile system in Chapter 6. We present a single cloud-based partitioning problem in Section 4.2.4 and we then extend it to the case of multiple clouds in Section 4.2.5. The assumptions are following:

- First, the cloud server is always reachable;
- Second, mobile devices are always connected to wireless network with symmetrical uplink and downlink bandwidths;
- Third, the energy consumption for transmitting data over the wireless interface in mobile devices is very close to the one for receiving data.

4.2.3 Integer Linear Programming

Integer-programming models arise in practically every area of application of mathematical programming to formulate problems such as capital budgeting, warehouse location, and scheduling. Suppose that the entire class of problems referred to as sequencing, scheduling, and routing are inherently integer programs. Consider, for example, the scheduling of students, faculty, and classrooms in such a way that the number of students who cannot take their first choice of classes is minimized. There are constraints on the number and size of classrooms available at any one time, the availability of faculty members at particular times, and the preferences of the students for particular schedules. Clearly, then, the ith student is scheduled for the jth class during the nth time period or not; hence, such a variable is either zero or one. Other examples of this class of problems include line-balancing, critical-path scheduling with resource constraints, and vehicle dispatching.

In binary problems, each variable can only take on the value of 0 or 1. This may represent the selection or rejection of an option, the turning on or off of switches, a yes/no answer, or offloading/non-offloading. A specialized branch and bound algorithm for solving BIPs, known as Balas Additive Algorithm can be employed to solve the ILP problem that requires the problem to be put into a standard form:

- The objective function has the form minimize $\sum_{j=1}^{n} c_j x_j$.
- The m constraints are all inequalities of the form $\sum a_{ij} x_j \geq b_i$ for i=1,2,...,m.

- All of the xj where j=1,2,...n are binary variables (can only have a value of 0 or 1).
- All objective function coefficients are non-negative.
- The variables are ordered according to their objective function coefficients so that $0 c_1\ c_2\ ... c_n$.

This seems to be a restrictive set of conditions, but many problems are easy to convert to this form. The program partitioning problem can be easily converted into the 0-1 or binary integer linear problem in the standard from stated above. However, integer variables make an optimization problem non-convex, and therefore far more difficult to solve. Memory and solution time may rise exponentially as you add more integer variables.

Since integer linear programming is NP-complete, many problem instances are intractable and so heuristic methods must be used instead. For example, tabu search can be used to search for solutions to ILPs. To use tabu search to solve ILPs, moves can be defined as incrementing or decrementing an integer constrained variable of a feasible solution, while keeping all other integer-constrained variables constant. The unrestricted variables are then solved for. Short term memory can consist of previous tried solutions while medium term memory can consist of values for the integer constrained variables that have resulted in high objective values (assuming the ILP is a maximization problem). Finally, long term memory can guide the search towards integer values that have not previously been tried. Other heuristic methods that can be applied to ILPs include hill climbing, simulated annealing, reactive search optimization, ant colony optimization, and neural networks to name a few. There are also a variety of other problem-specific heuristics, such as the k-opt heuristic for the travelling salesman problem. Note that a disadvantage of heuristic methods is that if they fail to find a solution, it cannot be determined whether it is because there is no feasible solution or whether the algorithm simply was unable to find one. Further, it is usually

impossible to quantify how close to optimal a solution returned by these methods is.

4.2.4 Single Cloud-Based Partitioning

We start with a single cloud-based partitioning solution Y that can be defined as $Y = \{y_0, y_1, ..., y_n\}$, where y_i denotes the partitioning result of the vertex i in the graph G constructed in Section 4.2.1. n is the total number of computational blocks such as method or object. The computation v_i is offloaded to the cloud if the value of y_i is 1, and v_i runs on the mobile device if the value of y_i is 0. The goal of the optimization problem formulated as 0-1 ILP (Refer to Section 4.2.3) is defined to find the optimal partitioning solution as follows,

$$\text{Miminize } \frac{T(Y,\mu)}{T_m} \cdot w + \frac{E(Y,\mu)}{E_m} \cdot (1-w), \tag{4.1}$$

subject to:

$$0 < T(Y,\mu) < T_m, \tag{4.2}$$

$$0 < E(Y,\mu) < E_m \tag{4.3}$$

where μ denotes current bandwidth of wireless network. w denotes the weight of tradeoff between execution time and energy consumption and set to 0.5. The weight can be selected statistically based on its history and type of applications or computation. We leave this weight selection issue for the future work. The first term of in Equation 4.1 is the total execution time for given partition Y and constraint μ divided by the local execution time T_m. $T(Y,\mu)$ represents the execution time via offloading. The second term of the equation is the total energy consumption for given partition Y and constraint μ divided by the local execution consumption E_m. $E(Y,\mu)$ represents the

energy consumption of remote execution E_m.

The objective function in Equation 4.1 requires two constraints: energy constraint and time constraint. The first constraint in Equation 4.2 indicates that the total execution time $T(Y, \mu)$ with offloading must not exceed the local execution time T_m. In the similar sense, the second constraint in Equation 4.3 indicates that the total energy consumption $E(Y, \mu)$ with offloading must not exceed the local execution time. The execution time $T(Y, \mu)$ is computed by,

$$T(Y, \mu) = \sum_{1 \in i \in n} \{t_i^m \cdot y_i + t_i^c \cdot (1 - y_i)\} + \sum_{1 \in i,j \in n} t_{i \to j} \cdot |y_i - y_j|, \tag{4.4}$$

where t_i^m denotes computation i running on the mobile device, and t_i^c denotes computation i running on the cloud. $t_{i \to j}$ denotes the cost of data transmission in time from mobile to cloud or vice versa. t_i^m, t_i^c, and $t_{i \to j}$ can be obtained at profiling time. t_i^m is only left when the offloading variable y_i becomes 1, or vise versa. We define the energy consumption for computation v_i as E_i, and the energy consumption for data transmission from vertex i to vertex j as $E_{i \to j}$. The E_i term goes to 0 when y_i becomes 0. This means that the offloading to the cloud only costs the energy for data transmission if $|y_i - y_j| \neq 0$. For the partition Y and current network bandwidth μ, we have

$$E(Y, \mu) = \sum_{1 \in i \in n} E_i \cdot y_i + \sum_{1 \in i,j \in n} E_{i \to j} \cdot |y_i - y_j| \tag{4.5}$$

4.2.5 Multiple Clouds-Based Partitioning

We now extend the single-cloud partitioning problem to the multi-cloud partitioning problem. For instance, consider two clouds: cluster $c1$ and $c2$. For given computation v, execution time $T_{c1}(v)$ running on $c1$ may be different from $T_{c2}(v)$ on the second

cluster $c2$. Energy consumption $E_{c1}(v)$ may be different from $E_{c2}(v)$. In similar sense, the cost of data transmission from mobile to $c1$ is most likely is the same as the one to $c2$.

We now extend the single-cloud partitioning problem to the multi-cloud partitioning problem by replacing $T(Y, \mu)$ in Equation 4.4 with the new $T(Y, \mu)$ in Equation 4.6. Denote multiple clouds $C = \{c_1, c_2, ..., c_k\}$ considered in,

$$T(Y, \mu) = \sum_{1 \in i \in n} \{t_i^m \cdot y_i + t_i^{\hat{c}} \cdot (1 - y_i)\} + \sum_{1 \in i,j \in n} t_{i \to j}^{\hat{c}} \cdot |y_i - y_j|, \tag{4.6}$$

where $t_i^{\hat{c}}$ represents computation i running on one of the multiple clouds among C. $t_{i \to j}^{\hat{c}}$ denotes the data transmission time from mobile to one of the multiple clouds C. An optimal cloud provider $\hat{c}$ among C is determined by metrics used in the profiler which we will discuss later in Equation 4.11. It helps simply the multi-cloud optimization problem by selecting one of clouds in advance and eliminates the cost of computing for solving the problem. Therefore, the new form of $T(Y, \mu)$ in Equation 4.6 stays very similar to the original form $T(Y, \mu)$ in Equation 4.4. We also redefine energy on a multiple-cloud formulation. Simply replace $E(Y, \mu)$ in Equation 4.5 with the new form $EE(Y, \mu)$ in Equation 4.7 to reformulate multi-cloud optimization problem.

$$E(Y, \mu) = \sum_{1 \in i \in n} E_i^{\hat{c}} \cdot y_i + \sum_{1 \in i,j \in n} E_{i \to j}^{\hat{c}} \cdot |y_i - y_j|, \tag{4.7}$$

Applying to our settings, we consider two clouds: one with Hadoop v1 cluster and the other with Hadoop v2 cluster. Compared to other VM-based approaches discussed in Section 1.1, our problem becomes much simpler as we consider resources in each Hadoop cluster as one equivalent resource in terms of performance in time and energy saving through it. It makes sense to make such assumption since the resource manager

in Hadoop assigns slots based on the size of memory in JVM (Java Virtual Machine), typically 256MB to 512MB per slot, 10-20 slots per machine [71].

4.3 Algorithms for Solving ILP

This section presents methods and approaches solving linear integer problems, developed during the last decades. These problems belong to the class of NP-hard optimization problems. To find out exact optimal solutions for this class of problems requires use of considerable computational resources. The development of efficient hybrid methods, combining in a suitable way the best features of different approaches (exact or approximate) is the actual direction, in which many researchers devote their efforts to solve successfully varioushard practical problems. The name linear integer programming refers to the class of combinatorial constrained optimization problems with integer variables, where the objective function is a linear function and the constraints are linear inequalities. The Integer Linear Programming (ILP or LIP) optimization problem can be stated in the following general form:

$$\text{Maximize } cx \tag{4.8}$$

$$\text{subject to:} Ax \leq b, \tag{4.9}$$

$$x \in Z^n, \tag{4.10}$$

where the solution xZ^n is a vector of n integer variables: $x = (x_1, x_2, ..., x_n)^T$ and the data are rational and are given by the $m \times n$ matrix A, the $1 \times n$ matrix c, and the $m \times 1$ matrix b. This formulation includes also equality constraints, because each equality constraint can be represented by means of two inequality constraints like those included in Eq. 4.10.

The linear integer programming problems are easier solvable than the convex non-

linear integer programming problems. A special case, 0-1 integer linear programming, in which unknowns are binary, is one of Karp's 21 NP-complete problems. An instance of Eq. 4.10 can be transformed in polynomial time to an instance of a 0-1 linear integer programming problem. But the 0-1 linear integer programming problem can be solved by a brute-force enumerative algorithm in $O(2^n poly(m,n))$ time [100]. On the other hand, an algorithm to solve the problem in time $O(2^{(1-s)n})$ for $s > 0$ when m is superlinear in n would contradict the Strong Exponential Time Hypothesis (SETH), which says that for every $s > 0$ there is a k such that k-sat cannot be solved in time $O(2^{(1-s)n})$ [96]. Williams gave an algorithm for 0-1 ILP that improves over exhaustive search even for a polynomial number of constraints [114]. The algorithm runs in time $2^{(1-s)n}$. For $s = \frac{1}{polylog(m)}$. Since s is subconstant, even for linear m, Williams result is not directly comparable to the result in [95]. Also note that a subconstant s does not contradict SETH. More recently, Impagliazzo et al. proposes an exact algorithm for the 0-1 Integer Linear Programming problem with a linear number of constraints that improves over exhaustive search by an exponential factor [95]. Their algorithm runs in time $2^{(1-poly(1/c))n}$ where n is the number of variables and cn is the number of constraints. The key idea for the algorithm is a reduction to the Vector Domination problem and a new algorithm for that subproblem. Under the Strong Exponential Time Hypothesis, this is qualitatively optimal in the sense that authors can only expect exponential improvement over exhaustive search if the number of constraints is linear. However, for the special case of formulas in conjunctive normal form the best algorithms achieve savings that are polylogarithmic in 1/c [97]. It is open if we can get the same for 0-1 ILP.

It should be noted, that there are many special cases (e.g. matching, node packing on appropriately restricted classes of graphs, and some matroid optimization problems) that belong to the class P of problems, solvable in polynomial time, i.e., there exist algorithms with polynomial time computational complexity, which can solve them. In general, the difficulty to solve (linear and/or nonlinear) integer programming problems

arises from the fact that unlike linear programming, for example, whose feasible region is a convex set, in integer programming problems, one must search for a lattice of feasible integer points to find an optimal solution. Unlike Linear Programming (LP) where, due to the convexity of the problem, we can exploit the fact that any local solution is a global optimum, the integer programming problems have many local optima and finding a global optimum to the problem requires one to prove that a particular solution dominates all the feasible points by arguments other than the calculus- based derivative approaches of convex programming with continuous variables. For this reason, the approximate algorithms solving ILP optimization problems are widely spread.

The following two sections briefly introduce exact methods (Section 4.3.1) and heuristics/population-based methods (Section 4.3.2) for solving ILP. In Section 4.3.1 the development of exact methods for solving LIP optimization problems is considered. It is divided in three subsections as follows: cutting planes approaches based on polyhedral combinatorics, enumeration techniques and relaxation and decomposition techniques. Section 4.3.2 is devoted to some heuristics and metaheuristic approaches, as well as to population-based evolutionary algorithms, designed to solve such class of optimization problems.

4.3.1 Exact Methods for solving ILP

There are, at least, three different approaches for solving integer programming problems, although they are frequently combined into hybrid solution procedures in computational practice ([98], [99], [102], [100], [101]): cutting planes algorithms based on polyhedral combinatorics, enumerative approaches and Branch-and-Bound, Branch-and-Cut and, Branch-and-Price methods; and Relaxation and decomposition techniques.

Cutting plane algorithms based on polyhedral combinatorics: The underlying idea of polyhedral combinatorics is to replace the constraint set of an integer programming problem by an alternative convexification of the feasible points and extreme rays of the

problem. Both the size and the complexity of the problems solved have been increased considerably when polyhedral theory was applied to numerical problem solving. A primal cutting-plane algorithm for general integer programs was proposed in [103]. A finitely convergent primal cutting-plane algorithm was proposed in [105], and simplified versions were published in [104], [106]. Because of poor computational experience, this line of research has been very inactive. An exception is a primal cutting-plane algorithm for the travelling salesman problem [107]. Although this algorithm has been moderately successful, it seems to be inferior to a fractional cutting-plane algorithm for the travelling salesman problem. Another strategy for cutting-plane algorithms is to maintain integrality and dual feasibility and then to use cuts to obtain primal feasibility. A finite algorithm of this type has been given by Gomory et al. [108]. Other similar algorithms have been proposed in [109], [110].

Enumerative approaches: These approaches are known under different names. The most popular of them are Branch-and-Bound, implicit enumeration and divide and conquer [100]. The explicit enumeration is the simplest approach to solving a pure integer programming problem by means of enumeration of all possibilities, which are finite in number. However, due to the combinatorial explosion of number of these possibilities resulting from the parameter size, only instances having relative small size could be solved by such an approach within a reasonable computational time limit. Sometimes many possibilities can be implicitly eliminated by domination or feasibility arguments. Besides straightforward or implicit enumeration, the most commonly used enumerative approach is called Branch-and- Bound (BnB), where the branching refers to the enumeration part of the solution technique and bounding refers to the fathoming of possible solutions by comparison to a known upper or lower bound on the solution value. The first BnB algorithm for general integer programs was introduced by Land and Doig [111]. The popularity of BnB approach increased substantially after the publication of BnB algorithm for the travelling salesman problem by Little et al. [112], because it demonstrated that large (at this time) problems could be solved by controlled enu-

meration. Balas gave the first implicit enumeration algorithm for general 0-1 integer programming problems [113].

Branch and cut: The bounds obtained from the LP-relaxations are often weak, which may cause standard BnB algorithms to fail in practice. It is therefore of crucial importance to tighten the formulation of the problem to be solved. The idea of dynamically adding the so called cutting planes to the problem is one way of obtaining stronger bounds. Combining the cutting plane algorithm with BnB results in the very powerful class of Branch-and-Cut (BnC) algorithms. The idea is to generate cutting planes throughout the BnB tree of a standard BnB algorithm, in order to get tight bounds at each node. The BnC algorithm consists of following major components: 1) automatic reformulation procedures, 2) heuristics which provide good feasible integer solutions and 3) cutting plane procedures which tighten the linear programming relaxation to the linear integer problem under consideration. These components are embedded into a tree-search framework as in the BnB approach to integer programming; whenever possible, there is used a fourth component: 4) the procedure permanently fixes variables (by reduced cost implications and logical implications) and does comparable conditional fixing throughout the search-tree. These four components are combined so as to guarantee optimality of the solution obtained at the end of the calculation. In some cases the algorithm may also be stopped prematurely to produce suboptimal solutions along with a bound on the remaining error. The cutting planes generated by the algorithm are facets of the convex hull of feasible integer solutions or good polyhedral approximations thereof and as such they are the tightest cuts possible. Lifting procedures assure that the cuts generated are valid throughout the search tree which aids the search process considerably and is a substantial difference to traditional (Gomory) cutting-pane approaches. For applications which use such Branch-and-Cut approach [114], [115], [116], [117] [118]. A direct outcome of these research efforts is that similar pre- processing and constraint generation procedures can be found in commercial software packages for combinatorial problems.

Branch and price: The philosophy of Branch-and-Price (BnP) is similar to the one of Branch-and- Cut. Indeed, the pricing and the cutting are procedures for tightening the LP-relaxation of the problem. In Branch-and-Price, the concept of column generation is combined with a Branch-and-Bound algorithm. The simplex algorithm arises at the origin from the column generation concept, where only variables with negative reduced costs are allowed to enter the basis at each iteration. Given a LP model with a huge number of variables, possibly depending exponentially on the instance size, it would be efficient to consider only the variables potentially improving the objective function. The main idea of column generation is to efficiently determine a variable with negative reduced costs to enter the basis, add it to the problem, resolve it and iteratively repeat this process until no variable with negative reduced costs exists anymore. In general, the method of Dantzig-Wolfe decomposition is often used for obtaining LP/LIP models with an exponential number of variables, which provide tighter bounds than the original compact LP/LIP pair. Description of Dantzig-Wolfe decomposition is given in [100]. An important point is that the column generation algorithm used must be aware of branching decisions and may only generate solutions respecting them. Another interesting question is whether the column generation algorithm should search for optimal solutions of the pricing problem or not. For a detailed review of column generation and BnP methods we refer to the recent book [119].

Relaxation and decomposition methods: There are three wide spread approaches for relaxation of the general LIP problem, which are designed to find an upper bound of the optimal value for the maximizing LIP problem: Linear Programming (LP) relaxation, Combinatorial relaxation and Lagrangian relaxation. The first two approaches extend the feasible domain without any change in the objective function of the problem. The third approach provides another maximizing objective function, which has the same or greater value in a fixed feasible domain. The now commonly used variable dichotomy scheme was proposed in [120]. The treatment of general upper-bound constraints by a division scheme together with an indexing scheme was introduced in [121]. The

sets considered are called specially ordered sets. This terminology is now widely used and the concept is very important in the global maximization of the piecewise linear nonconcave functions. Beale and Forrest developed this approach which enables the implementation of the division scheme without the explicit use of auxiliary integer variables [122].

For realization of the combinatorial relaxation there are at least two approaches exploiting the combinatorial structure of the problem. The first approach is based on the concept of valuated matroids, introduced by Dress and Wenzel [123]. Greedy-type algorithms can be used for optimization. The other approach, which is called the structural approach, utilizes algorithms to compute an upper bound on the objective function and is often based on a graph-theoretic method [124].

4.3.2 Heuristics and population-based methods for solving ILP

Since the integer programming optimization problems and the LIP optimization problems in Eq. 4.10, as mentioned above, belong to the class of NP-hard optimization problems, it is very difficult and requires great computational efforts to find out an optimal and even a feasible solution for large size problems. Very often it is more important an acceptable solution to be found out, instead to wait a long time to obtain the optimal solution. Some flexible constraints may exist in the description of the problem model and they could be changed only a little bit. The exact algorithms need to resolve the problem even in case of a little change of one constraint. This may be very time consuming and could be expensive for real applications. The approximate algorithms are not so sensitive to little changes in some constraints. Some of them solve the problem consecutively, while it is decomposed into parts. In such case the resolving of the entire problem is not necessary. The approximate algorithms as subroutines in the exact algorithms find a broad field of application. They could be used to find out a suitable initial solution or to tighten the feasible domain of solutions and to direct the search for an optimal solution. A huge number of approximate algorithms

has been created for the solution of large real life LIP optimization problems without any guarantee for optimality of the final solution [125]. According to the quality of the solutions obtained, the approximate algorithms can be divided into three groups as follows: (1) approximate algorithms having arbitrary predetermined accuracy (absolute or relative); (2) approximate algorithms having in advance determined accuracy, whereat the approximation error does not tend to zero; (3) heuristic algorithms in this case it is supposed on the base of experiments and other evaluations, that with great possibility they will find out a solution of the problem with good quality using reasonable computational resources, but there is not available any guaranteed mathematical evaluation of their accuracy. The development of approximate algorithms, for which it has been theoretically proven, that they terminate their performance using a polynomial number of standard mathematical operations, is especially important. The basic heuristic strategies, used in the approximate algorithms could be considered as: (4) constructive algorithms and (5) local-improvement algorithms.

The most familiar and powerful metaheuristics are Simulated Annealing ([126], [127], [128]), and Tabu Search ([129], [130]). They are based on Local Search techniques [131]. Other well-known approaches in this group are Guided Local Search [132], Iterated Local Search [133] and Variable Neighbourhood Search [134]. The Population-based algorithms are a large group of metaheuristics based on the natural practices of surviving of the best that have a learning capability. These include: Genetic Algorithms ([135]), Scatter Search ([136]), Ant Systems/Ant Colony Optimization ([137]), Particle Swarm Optimization ([138]) and Memetic Algorithms [139].

4.4 Profiling

There are several history-based approaches in profiling hardware and mobile applications. CloneCloud [16] proposes a dynamic profiler is responsible for the collection of cost metrics data that facilitates in the development of cost models, used for offloading

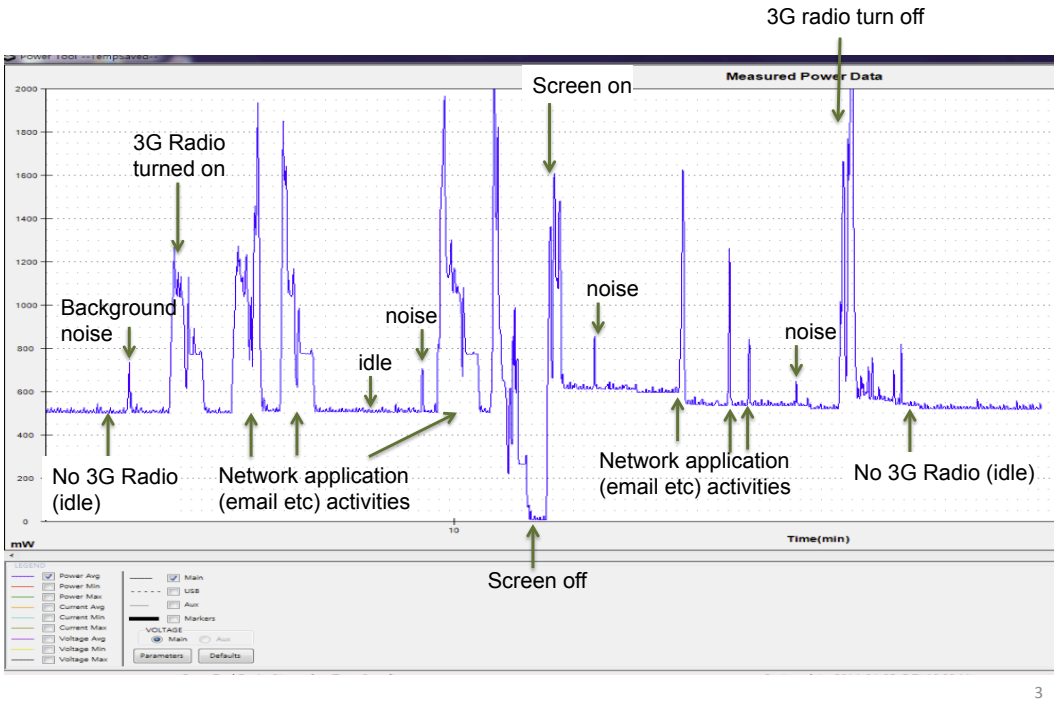

Figure 4.2: Time-series measurement on energy consumption of a mobile device (Samsung Galaxy S3) using the Monsoon power monitor [38].

decision making. The cost metrics can be obtained by running the executable repeatedly on both sides (mobile device, cloud) with different input settings. MAUI [14] uses a profiler (optimization engine) that analyzes energy consumption involved in the local and remote execution of the code. Moreover, MAUI profiles offload methods and use history-based approach to predict the execution time of a particular code. The profiler collects information regarding the application energy consumption and data transfer requirements. ThinkAir [18] supports three profilers that coordinate with the energy model. The device profiler monitors the energy consumption of the device hardware resources, such as processor, antennas, display screen etc. The program profiler monitors the program parameters, for instance, execution time, acquired memory, thread CPU time, number of instructions and method calls. The network profiler monitors network related parameters, for instance, bandwidth, connectivity, and delay.

Figure 4.2 presents the energy consumption on several hardware, software, and net-

work events: 3G/4G-LTE radio on/off, network application (email, Facebook) events, screen on/off. For instance, LCD screen consumes relatively large energy even when 3G/4G-LTE radio is on. Interestingly, we observe that the activation of each hardware component often leads to hit the peak energy consumption for a short period of time. Therefore, the energy consumption of software, hardware, and network cannot be separately profiled. We list up hardware components that significantly affect energy consumption: WiFi, 3/4G-LTE radio, GPS, screen, and Bluetooth. In profiling time, we carefully control turning on and off such components for accurate measurement of energy.

$$\arg\min_{\hat{c}\in C}\{\frac{(t_i^{\hat{c}} + t_{i\to j}^{\hat{c}})}{t_i^m} \cdot w + \frac{(E_i^{\hat{c}} + E_{i\to j}^{\hat{c}})}{E_i} \cdot (1-w)\} \tag{4.11}$$

We now continue to define the optimal cloud selection $\hat{c}$ based on time and energy already defined in previous section, but with normalized notation with a bar. In order words, $t_i^{\hat{c}}$ denotes the normalized execution time for computation i, and $t_{i\to j}^{\hat{c}}$ denotes data transmission time from mobile to cloud. In similar sense, energy $E_i^{\hat{c}}$ for computation and the one for data transmission $E_{i\to j}^{\hat{c}}$ are presented. According to Equation 4.11, we obtain the optimal cloud choice to minimize the overall cost of time and energy. By default, we set w to 0.5. This will be fed into Equation 4.6 and Equation 4.7.

4.4.1 Energy Model

There are several approaches in modeling energy consumption across multiple research areas: averaging model [40] [39]; regression model [41] [42]; probabilistic model [44] [45]; machine learning model [43] [46]; classification model [47]. Among aforementioned approaches, PowerTutor [41] only matches our requirements for energy modeling on smartphones with *on-board sensors* and no external power monitor requirement. ThinkAir [18] modifies the PowerTutor model to add additional hardware (GPS, audio interface) pro-

filing capability. The major drawback for that are the inability to adapt to changes in energy consumption. Frincu et al. [48] states that the energy consumption changes over time from the analysis of 190 industrial consumers dataset. We leave this capability of adapting the energy model for the future work although it seems to be very important. For simplicity, we adopt the approach in ThinkAir [18] by extending PowerTutor [41] to additional capabilities to taking into account GPS, audio, more hardware components in our mobile devices.

CHAPTER 5

Data-Intensive MCC Application: A Case of Automatic Speech Recognition

This chapter introduces the concept of automatic speech recognition as one of the representative mobile application of data-intensive computing. The organization of the chapter includes three subsections: introduction to automatic speech recognition (ASR), its computation for personalization, and finally related work.

5.1 Introduction

An automatic speech recognition (ASR) becomes very popular in these days. In Figure 5.1, we briefly describe the overall procedure of ASR: from feature extracting, phone sequence recognition to word sequence recognition. For readers who are more interested in details, we recommend to read Rabiner's book [24]. Big companies such as Samsung, Google, Yahoo, Facebook, and Amazon know enough about users from previous contacts, purchase history, and other sources to structure special offers and purchase recommendations well suited to your tastes. For instance, Amazon uses collaborative filtering to determine what music or books to recommend. Such efforts can be seen as personalizing their services to improve a factor of user satisfaction in terms of recommendation, recognition, estimation, and prediction. In this paper, we limits ourselves to the context of the automatic speech recognition service as a representative MCC service that can benefit from personalization on top of big data computing.

With the rising attention on the voice-enabled service, the personalization of such

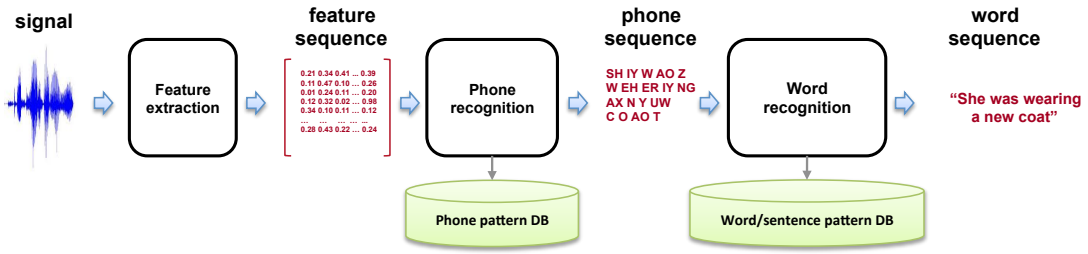

Figure 5.1: An overview of automatic speech recognition.

service is preferred. One direction to provide more personalized nature to ASR is known as speaker adaptive recognition or speaker adaptation, where an acoustic model is adapted to an individual speaker to improve the quality of speech recognition. The other direction is to personalize a language model in ASR, where a statistical language model assigns a probability to a sequence of words according to individual speaker's way of speaking reflecting direct, tone, intonation, and other linguistic properties that may be uniquely identifiable among multi-user ASR environment. This work focuses more on the former while exploring both of directions to provide better personalized ASR experience. One research question is whether or not the speaker adaptation really works for mobile services in case, where each utterance is relatively shorter than the other scenario such as continuous transcriber, web search. An example would be *call mom*, *search restaurant nearby*. This shorter utterance pattern may not guarantee the improvement of a personalized acoustic and language model at all times. Another research question is whether or not the small amount of user's speech data with no large computation requirement (significantly less than typical model training) can actually contribute to making the model better.

The idea of the personalizing ASR has been extensive discussed in the literature ([78], [77]). We however realize that implementing such system is a big challenge due to a lack of scalable, distributed computing and storage systems. Often, researchers in the area of ASR assume that a scalable system that supports large-scale ASR system is readily available when needed. Furthermore, a trend toward personalizing acoustic model and language model boosts this problem even more difficult in terms of paral-

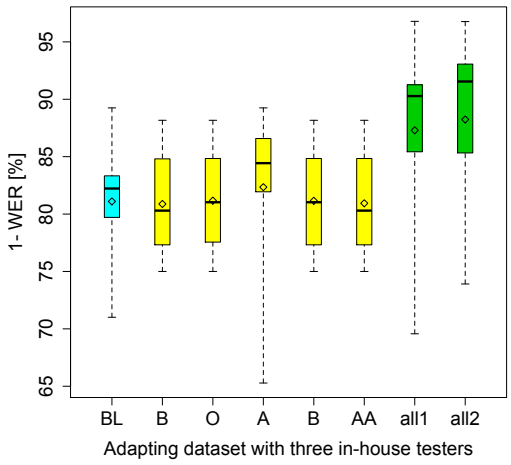

Figure 5.2: The feasibility test on speaker adaptation by personalizing acoustic models with children's books and three testers. We size the adaptation set from 10-30 to 89 utterances.

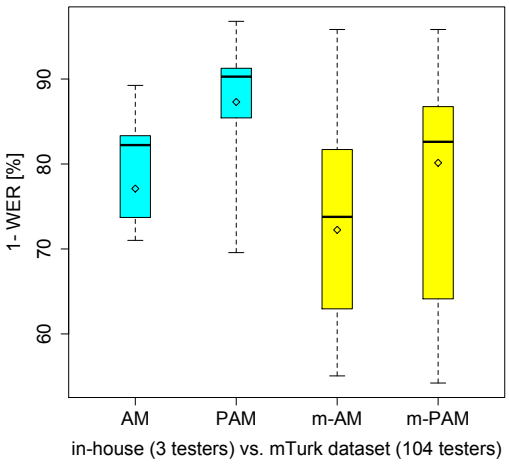

Figure 5.3: The feasibility test on speaker adaptation by personalizing acoustic models with smartphone scenario collected from Amazon Mechanical Turk workers. We fix the adaptation set in 200 utterances including daily voice service use cases such as call mom.

lel computation and distributed storage. Parallel computing must support personalized model generation of individual users to meet SLAs in voice-enabled services. Distributed storage must store user history data (e.g., speech files, transcribed text, other user history logs, acoustic models, language models) in a virtually infinite capacity. For instance, CMU Sphinx [65] requires 10 mega bytes (MB) of an acoustic model and 30MB of a language model per person in our personalized ASR scenario. In order to personalize at least 100 million users in Samsung's S-Voice service, the required storage capacity only for acoustic models can easily exceed over 4 petabytes (PB). In fact, each user requires 62MB in total, becoming at lease 6.2 PB in the same scenario that cannot be dealt with small-scale computing and storage systems. Thus, the simple term, personalization of the ASR service challenges in several ways: large-scale computation, distributed storage, and reliable and seamless data sharing between compute nodes and storage's data nodes.

The idea of the personalizing ASR has been extensive discussed in the literature ([78], [77]). We however realize that implementing such system is a big challenge due to a lack of scalable, distributed computing and storage systems. Often, researchers in the area of ASR assume that a scalable system that supports large-scale ASR system is readily available when needed. Furthermore, a trend toward personalizing acoustic model and language model boosts this problem even more difficult in terms of parallel computation and distributed storage. Parallel computing must support personalized model generation of individual users to meet SLAs in voice-enabled services. Distributed storage must store user history data (e.g., speech files, transcribed text, other user history logs, acoustic models, language models) in a virtually infinite capacity. For instance, CMU Sphinx [65] requires 10 mega bytes (MB) of an acoustic model and 30MB of a language model per person in our personalized ASR scenario. In order to personalize at least 100 million users in Samsung's S-Voice service, the required storage capacity only for acoustic models can easily exceed over 4 petabytes (PB). In fact, each user requires 62MB in total, becoming at lease 6.2 PB in the same scenario that cannot be dealt with small-scale computing and storage systems. Thus, the simple term, personalization of the ASR service challenges in several ways: large-scale computation, distributed storage, and reliable and seamless data sharing between compute nodes and storage's data nodes.

To this end, personalizing voice-enabled service becomes a MCC system problem for given well-known speech libraries, where a thin mobile client offloads heavy computation to clouds with big data. This trend behooves us to consider the big data platform as a candidate for the proposed ASR backbone that has personalization capability in a scalable manner for from millions to billions of users. Hadoop [71] is on the top list since it has been deployed in several large-scale services, (e.g., Yahoo reported 10s of thousands machines in their Hadoop cluster, Facebook reported 600 machines in their cluster [80]). In our experiments, our Hadoop cluster (59 machines) is able to create one million users' personalized acoustic models (PAMs) within one hour and 100 mil-

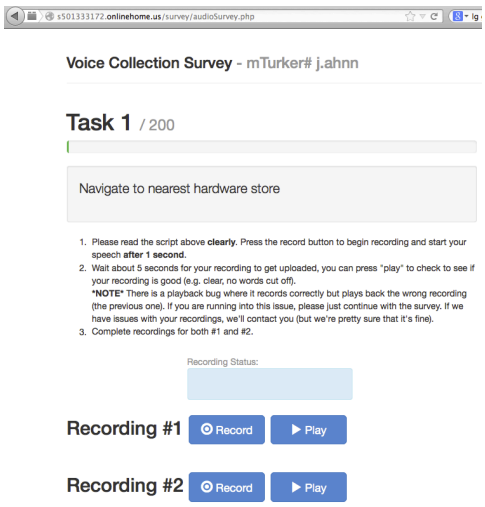

Figure 5.4: We developed voice recording Web pages based on WIMI toolkit [49] to be linked from Amazon mechanical Turk.

lion PAMs within 3.5 days without applying any optimization techniques. This means the voice-enabled system periodically can refresh speaker-adapted acoustic models in an every hour for a one million active user scenario, where inactive user's PAM does not require to be renewed frequently. Furthermore, the integration with the big data infrastructure for such voice-enabled services allows providing many more potential opportunities in computation (scalable computing environment and storage with reliability), analysis with eco-system tools (e.g. elasticsearch, hive, pig, etc.).

5.2 Personalization of ASR: Speaker Adaptation

The applied system architecture is redrawn in Figure 5.5. A MCC application for automatic speech recognition running on top of BigMobile framework is developed as shown in Figure 5.6. A further screenshot for Web-based ASR application is presented in Figure 5.7. Through personalization of ASR, the word error rate is enhanced from 75% to 83.33%. We first start with three male testers: one native American and two

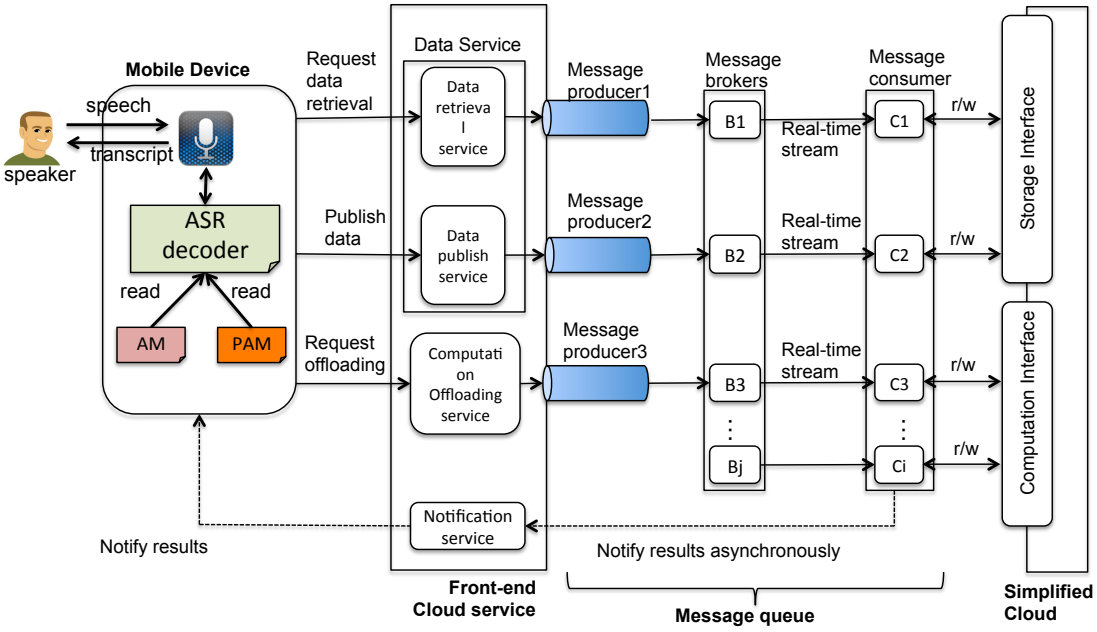

Figure 5.5: An applied BigMobile system architecture for the automatic speech recognition mobile application. Refer to Figure 2.1 for completeness as the cloud system is oversimplified.

testers who have lived in U.S. more than 10 years. Each tester recorded one script twice so that the first recording is used for adapting and the second for testing, where one script makes around 10-30 utterances using Sphinx. Figure 5.2 presents the accuracy on PAM in 1-(word error rate) based on scripts from five different children's books (yellow-coded). The bar plot plots min, 25 percentile, median, 75 percentile, max from bottom to top. The dot within the bar indicates mean of the distribution. The blue-colored baseline BL (with general acoustic model) on the first column is presented to compare with other PAMs. The last two columns present the accuracy of all script combined: all1 denotes the first recording is used for adaptation; all2 denotes the second recording is used for adaptation. With the small adaptation set of 10-30 utterances (B, 0, A, B, AA), it is difficult to observe the improvement due to larger variations in accuracy. We however clearly see the benefit of PAM in all1 and all2 dataset, which consist of 89 utterances each. The results say that we can have well-known accuracy improvement (10%) with around 90 utterances in children's books, compared to the

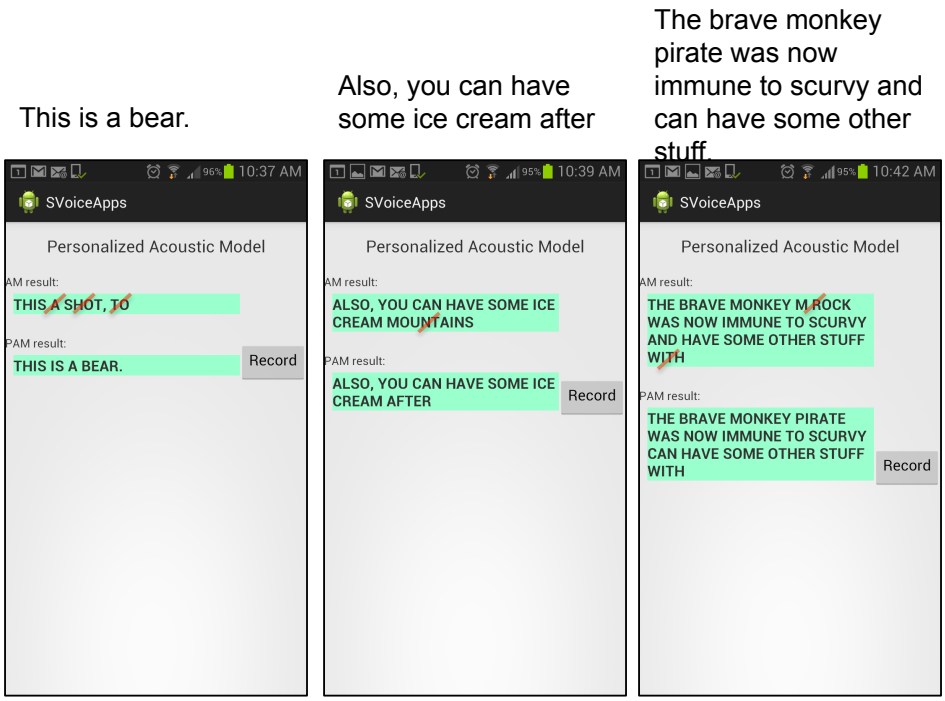

Figure 5.6: Screenshots for an automatic speech recognition mobile application with original script on the top of each figure.

baseline (BL).

In order to better verify the speaker adaptation, we reflect smartphone user's use cases to scripts such as *call mom*, *search for best gifts for thanks giving*, *directions to home*, and *best Japanese restaurant near me*. Scripts consisting of 200 utterances are sampled from Samsung's S-Voice logs and survey from 200 Samsung employees via surveymonkey.com to make it more realistic. Figure 5.4 presents voice collection Web pages we developed to be linked to an Amazon mechanical Turk post. Figure 5.3 compares 3 testers' case (AM, PAM) with 104 testers' data set (m-AM, m-PAM) collected from mTurk from Nov. 8, 13 to Nov. 19, 13, where each mTurk worker is given 7 to 8 dollars for twice of 200 utterance recordings. AM denotes a case for applying a general acoustic model, and PAM denotes a case for applying a personalized acoustic model. Cases starting with m- are based on mTurk data set. Although we observe lots of ambient noise on mTurk workers' recordings, the results of speaker adaptation seem to be as compatible as it claimed earlier in testers (see Figure 5.2).

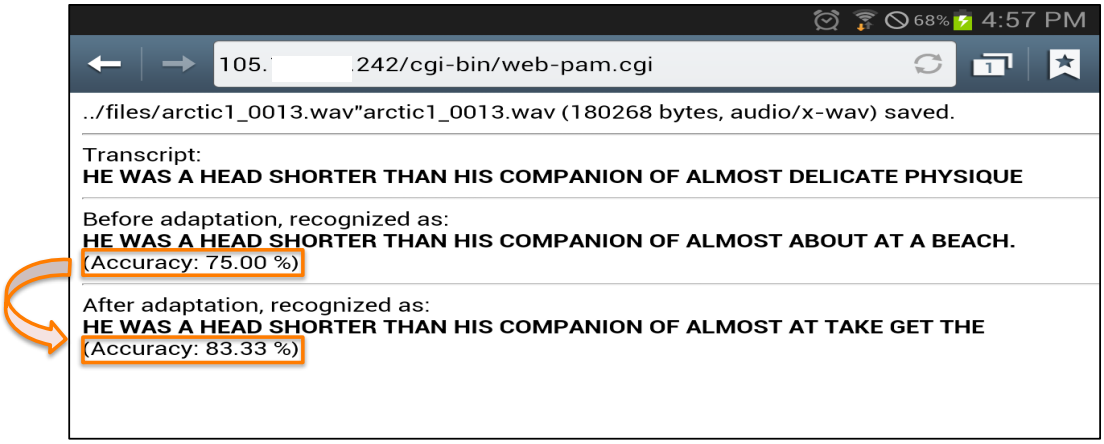

Figure 5.7: Screenshots for an automatic speech recognition Web application.

Figure 5.3 presents AM and PAM for each of three testers data and mTurk data set, and the speaker adaptation works well (10% and 9% improvement on average, respectively) in both of cases although mTurk data set is overall degraded the speech recognition accuracy due to uncontrolled recording environments. In the next section, we will discuss how the personalized ASR system can be designed and implemented in any speech libraries such as Sphinx, Kaldi, and HTK.

Sphinx libraries [65] consists of six different jobs called $sphinx\_fe$ in SphinxBase 0.8, $pocketsphinx\_mdef\_convert$ in PocketSphinx 0.8, bw, $mllr\_solve$, map_adapt, and $mk\_s2sendump$ that must be defined in one workflow with dependencies since an output of prior results is fed to the next program in order. The last four programs are originated from SphinxTrain 1.0.8. The first four programs generate a personalized acoustic model by applying the MLLR method while the last two programs generate another personalized acoustic model by applying the MAP method. Details of methods are out of scope and refer to [65]. We start with a bundle acoustic model in PocketSphinx called $hub4wsj\_sc\_8k$ for our result to be re-playable by any other researchers. An automated script creates two different PAMs from MAP and MLLR method. From the result, we take a model that performs better speech recognition.

5.3 Related Work

Sphinx is a well-known speech library written in C and Java and developed by CMU. One drawback to this is no neural network support. Kaldi [68] implements advanced features such as subspace GMM and FST-based speech recognizer. Unlike Sphinx [65] and HTK [67], it is written in C++ instead of C. Another interesting part of Kaldi is that it is using weighted finite state transducer (WFST) as the unifying knowledge source representation. However, the composed decoding WFST would naturally outgrow the system memory as the vocabulary size goes large and knowledge source gets more complicated, . HTK is another speech library that is mainly based on MPE and MMIE. Julius is a high-speed speech recognizer that can decode a 60k vocabulary. One speed-up techniques of Sphinx 3.X was context-independent phone Gaussian mixture model selection (CIGMMS), however, Julius [69] comes with a set of Japanese models, not English, and this might be one of the reasons why it is not as popular as HTK, Sphinx, Kaldi. In the consideration of large-scale speech-enabled services on top of the big data infrastructure such as Hadoop, all four speech libraries aforementioned require lots of efforts on deployment in service by even requiring source code modification. The reason is that speech data and related logs are stored in a distributed data platform from which speech library needs to retrieve data to process and provide services such as speech-to-text transcription and personalized asr service requiring further online and offline pre/post processing. In order to implement scalable system and advanced ASR system, seamless integration with the data infra is unavoidable.

Typical ASR services are Samsungs S-Voice and Apples Siri. They are all based on server-side speech recognition engines that may or may not provide personalized models due to non-disclosure of their technologies. The common techniques in the speaker personalization have been extensively discussed in this area ([77], [78]). Some work is based on acoustic models while some other work is based on language models. Our work focuses more on the acoustic model while considering language model personal-

ization since the speaker adaptation required generating a language model for adapting data set to be eventually applicable to the personalized language model.

Recently, the study on distributed speech recognition system has been extensively explored in the literature [64]. You et al. [63] proposes an OpenMP-based speech recognizer for continuous speech streams. The drawback of their proposal is that the system keeps all the streaming data in memory, and potential out-of-memory problem exists for large data stream. Chong et al. [62] proposes a method to parallelize speech recognizer for parsing given voice streams. No proposal is made for scalable system as data go bigger, where scalable storage and computing need to be in place. Chang et al. [50] presents a Cloud-assisted speech recognition service for personal mobile device. Even though the scheme is highly optimized for mobile devices, back-end data storage and computation nodes are not scalable, specially when highly personalized services are needed such as personalized- acoustic model and language model generation/retrieval in a real time manner.

Therefore, we believe that the introduced ASR can be a representative example of data-intensive MCC application. The evaluation of ASR running on top of BigMobile is presented in Section 7.2.

CHAPTER 6

Data-Intensive Computational Cloud

The data-intensive computational cloud in BigMobile consists of several components to achieve asynchronous offloading, server-side fault-tolerant and scalable offloading message delivery, and big data platform integration. As depicted in Figure 2.1, there are three major components in BigMobile clouds as follows,

- First, the reliable offloading message queue consists of multiple publishers, brokers, and consumers so that it can provides a reliable and scalable mechanism within offloading communication between the mobile and the cloud (refer to Section 6.1.
- Second, the front-end cloud service contains a data service (Section 6.2), offloading service (Section 6.3), and notification service.
- Third, cloud resources provide the fuse to convention big data platform Hadoop. The fuse includes two interfaces: a computation interface to access to computational resource (Section 6.4), and a native cloud storage interface to access to cloud storage such as Hadoop distributed file system and other database running as Hadoop-ecosystem.

This chapter is organized as follows: the overall system architecture for data-intensive cloud support including reliable message queue, RESTful data service, and RESTful computation service, and finally computation offloading service.

6.1 Reliable Message Delivery Queue

The BigMobile's message queue (BMQ) aims at being scalable, reliable, but high-performing in a distributed design. We believe that offloading requests delivered by producers have to be transparently routed to all the interested sink, corresponding to consumers. To foster system scalability and message availability, BMQ aims at two aspects. First, offloading message production and consumption should be possible at different times for time decoupling. Second, sinks are not necessarily know each other for space decoupling. In other words, communication has to be asynchronous and anonymous among producers and consumers as in traditional publish/subscribe systems. BMQ is a distributed, partitioned, and replicated commit message delivery service. It provides the functionality of a messaging system, but with a set of unique design goals in four folds,

- BMQ maintains feeds of messages in categories called topics;
- BMQ calls processes that publish messages to topic producers;
- BMQ calls processes that subscribe to topics and process the feed of published messages consumers;
- BMQ runs as a cluster comprised of one or more servers each of which is called a broker.

The basic system architecture of the message queue is depicted in Figure 2.1. At a high level, producers for delivering offloading requests forward messages over the network to the BMQ cluster (consisting of multiple brokers), which in turn serves them up to consumers. Communication between BMQ entities (consumers, brokers, and producers) should interact with a simple and reliable TCP protocol. Among several open source projects such as RabbitMQ [81], Kafka [82], Kafka seems to be promising as it's stability, reliability, and scalability are already proven by many big data companies like Google, Yahoo, and Facebook.

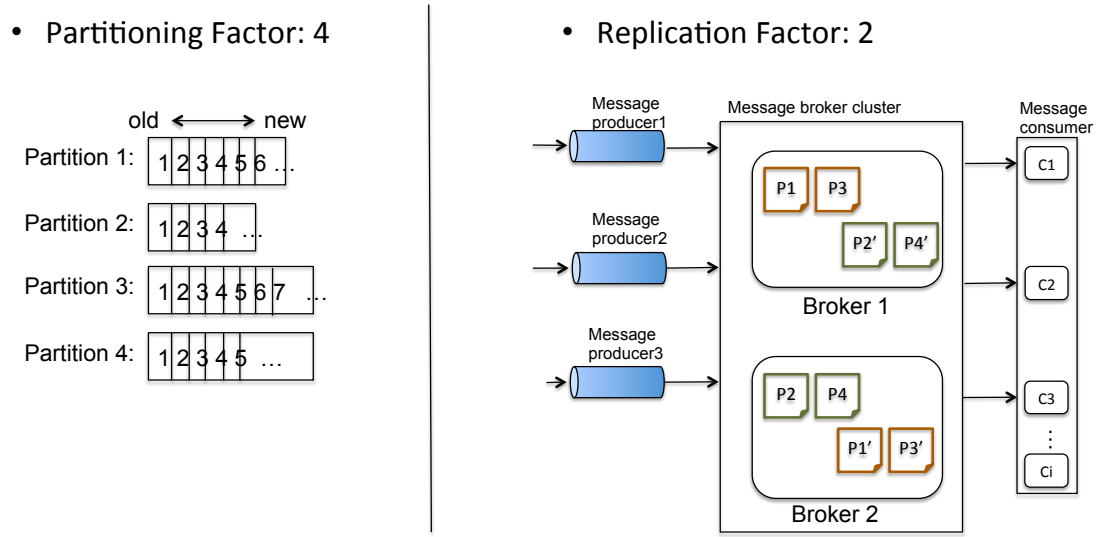

Figure 6.1: Illustration of BigMobile's message queue with partitioning (factor=4 on the left) and replication (factor=2 on the right).

A topic is a category or feed name to which messages are published in Kafka. For each topic, the Kafka cluster maintains a partitioned log [82]. Our BMQ can be seen as one of the kafka applications, and it exploits its topic model so that the message queue system can recognize and separate both offloading requests and data service requests at the same time. In our BMQ design, we employ the feature of parallelism in Kafka so that BMQ is able to provide both ordering guarantees and load balancing over a pool of consumer processes by the partition within two different service/topic types.

As shown in Figure 6.1, BMQ provides following guarantees,

- Both offloading and data requests sent by a set of producers to a particular service partition will be appended in the order they are sent. That is, if a request $r1$ is sent by the same producer as a request $r2$, and $r1$ is sent first, then $r1$ will have a lower offset than $r2$ and appear earlier in the log.

- A set of consumer instances sees requests in the order that they are stored in the log.

- For a topic with replication factor N, we will tolerate up to $N - 1$ broker server

failures without losing any messages committed to the log.

Figure 6.1 illustrates how BigMobile's message queue works with partition and replication. With the factor of partition factor 4, incoming messages to the message queue is automatically splited into four different chunks in order. The newer messages are located on the right-hand side. With the replication factor 2, the messages are stored in two different machines with the granularity of partitions. Broker 2 can survive and provide messages to consumers even when broker 1 fails for some reason since the copy of partitions are kept in the survived broker.

6.2 RESTful Data Service

The data service provides two functionalities: data upload and data retrieval. The RESTful APIs allow for application developers to interact with BigMobile's cloud back-end by a simple HTTP(s) interface. We implement RESTful APIs using Jetty [79] that is one of the commonly used packages. The data service constitutes two subcomponents: offline storage and online storage.

6.2.1 Offline Storage

The offline storage is a key part in the big data platform due to the high volume of files. For instance, Facebook currently stores 20 PB [70]. Petabytes-scale offline storage must be supported in a distributed manner where the file system maintains three replicas of data for fault-tolerance. We first consider a distributed file system supported by Apache Hadoop v1 (HDFS) due to its proven stability while unstable Hadoop v2 is in town. The major bottleneck of Hadoop v1 is known to be Hadoop Distributed File System (HDFS). We will investigate Hadoop v2 in our evaluation chapter since Hadoop v2 claims the improvement of the file system. We implement our system in both Hadoop versions to compare the significance of performance.

6.2.2 Online Storage

The online storage is known to be faster than the offline storage system in nature. HBase [72] is one of the best key-value *data storage* when we develop the system. For the personalized ASR application, we require 1KB-100KB files including logs and speech files, 10MB for personalized language model, and 30MB for personalized acoustic model. Due to the nature of personalization, the system must fetch files fast in real time. In order to support such various files, the online storage such as HBase has a fundamental drawback when the size of file goes big since the online storage is designed not for large files but for small data or small files . To resolve this issue in a systematic way, there are several recently proposed systems such as Facebook's Haystack [70], Twitter's Blobstore [73], and Hbase large object (LOB) storage [74]. They are very similar to our *object storage* and *data storage* in a way that all three systems aim to store small and large binary large objects, and rapidly fetch files at low cost. Our object storage supports list, create and delete buckets, and put and get objects via the RESTful APIs implemented by Jetty [79].

The data storage is a key-value based data store where HBase can be accessed by RESTful APIs, and its supporting operations are create tables, store/delete/update values in tables, get a rows in table. The APIs allow the BigMobile system to easily store and fetch data from tables. On top of the *data storage*, we implement the *object storage* for storing large objects such as PAM files. The data storage is used both in real-time (speech-to-text conversion) and batch processing (individual user's PAM generation) from the execution engine. The design principle of the *object storage* is quite similar to Facebook's Haystack where the data storage is a major storage for file's metadata and larger files are stored in the HDFS part of the object storage. If a fetching file is less than a threshold, say 100KB, it reads a pointer to HDFS from the metadata in the data storage that leads to the real file in HDFS. We note that the threshold is somewhere we need to investigate further in the evaluation section.

6.2.3 RESTful APIs Specification

The RESTful APIs provide a simple set of commands *PUT, GET, DELETE* for easy data access operations. We describe the format of request, response, and object format accordingly.

Request Format. For *PUT* requests, the request body must be JSON, with the Content-Type header set to application/json. Authentication is done via HTTP headers. The BigMobile-Application-Id header identifies which application you are accessing, and the BigMobile's REST-API-Key header authenticates the endpoint.

Response Format. The response format for all requests is a JSON object. Whether a request succeeded is indicated by the HTTP status code. A 2xx status code indicates success, whereas a 4xx status code indicates failure. When a request fails, the response body is still JSON, but always contains the fields code and error, which you can inspect to use for debugging. For example, trying to save an object with invalid keys will return the message:

Object Format. Storing data through the RESTful API is built around a JSON encoding of the object's data. This data is schemaless, which means that you don't need to specify ahead of time what keys exist on each object. You simply set whatever key-value pairs you want, and the backend will store it. Keys must be alphanumeric strings. Values can be anything that can be JSON-encoded. Each object has a class name that you can use to distinguish different sorts of data. For example, we could call file objects.

Listing 6.1: An example of RESTful APIs supported by BigMobile's data service

```
1 // 1.1. Creating Objects
2 curl -X PUT \
3   -H "BigMobile-Application-Id: ${APPLICATION_ID}" \
4   -H "BigMobile-REST-API-Key: ${REST_API_KEY}" \
5   -H "Content-Type: application/json" \
6   -d '{"userName":"Joey Ahnn","age":30}' \
```

```
  https://172.28.48.31/dataservice/UserObject

// 1.2. Response to Creating Objects
Status: 201 Created
Location: https://172.28.48.31/dataservice/UserObject/PbednuqPvsm

// 1.3. JSON formatted Body of the response to 1.1
{
  "createdAt": "2014-08-20T02:06:57.931Z",
  "objectId": "PbednuqPvsm"
}

// 2.1. Retrieving Objects
curl -X GET \
  -H "BigMobile-Application-Id: ${APPLICATION_ID}" \
  -H "BigMobile-REST-API-Key: ${REST_API_KEY}" \
  https://172.28.48.31/dataservice/UserObject/PbednuqPvsm

// 2.2. JSON formatted Body of the response to 2.1
{
  "userName": "Joey Ahnn",
  "age": "30",
  "createdAt": "2014-08-20T02:06:57.931Z",
  "updatedAt": "2014-08-20T02:06:57.931Z",
  "objectId": "PbednuqPvsm"
}

// 3.1. Updating Objects
curl -X PUT \
  -H "BigMobile-Application-Id: ${APPLICATION_ID}" \
  -H "BigMobile-REST-API-Key: ${REST_API_KEY}" \
  -H "Content-Type: application/json" \
  -d '{"age":31}' \
  https://172.28.48.31/dataservice/UserObject/PbednuqPvsm
```

```
41
42 // 3.2. JSON formatted Body of the response to 3.1
43 {
44   "updatedAt": "2014-08-21T18:02:52.248Z"
45 }
46
47 // 4.1. Deleting Objects
48 curl -X DELETE \
49   -H "BigMobile-Application-Id: ${APPLICATION_ID}" \
50   -H "BigMobile-REST-API-Key: ${REST_API_KEY}" \
51   https://172.28.48.31/dataservice/UserObject/PbednuqPvsm
```

Listing 6.1 presents an example code for putting objects to the cloud storage and for fetching them from the cloud storage through RESTful APIs. (1.1, line 2) To create a new object on the cloud storage, send a *PUT* request to the class URL containing the contents of the object. (1.2, line 10) When the creation is successful, the HTTP response is a 201 Created and the Location header contains the object URL for the new object. (1.3, line 14) The response body is a JSON object containing the objectId and the createdAt timestamp of the newly-created object. (2.1, line 20) Once one created an object, she can retrieve its contents by sending a *GET* request to the object URL returned in the location header. (2.2, line 26) The response body is a JSON object containing all the user-provided fields, plus the createdAt, updatedAt, and objectId fields: (3.1, line 35) To change the data on an object that already exists, send a PUT request to the object URL. Any keys you don't specify will remain unchanged, so you can update just a subset of the object's data. (3.2, line 43) The response body is a JSON object containing just an updatedAt field with the timestamp of the update. (4.1, line 48) To delete an object from the cloud storage, send a DELETE request to its object URL.

6.3 RESTful Computation Service

The definition of RESTful APIs stays very similar to the data service as described in Section 6.2. Listing 6.2 presents an example code for submitting an offloading request to the cloud computation service through RESTful APIs. It provides a command *PUT* for submitting computation to the cloud. We describe the format of request and response accordingly.

(1.1, line 2) To create an offloading request, send a PUT request to the class URL containing the contents or parameters of the request. (1.2, line 10) When the submission is successful, the HTTP response is a 201 Created and the Location header contains the request URL for the new job, which can be used in tracking the submitted job for its lifetime. (1.3, line 14) The response body is a JSON object containing the objectId and the createdAt timestamp of the newly-created offloading request or object.

Listing 6.2: An example of RESTful APIs supported by BigMobile's cloud computation service

```
1  // 1.1. Creating a offloading request to cloud
2  curl -X PUT \
3    -H "BigMobile-Application-Id: ${APPLICATION_ID}" \
4    -H "BigMobile-REST-API-Key: ${REST_API_KEY}" \
5    -H "Content-Type: application/json" \
6    -d '{"OffloadingMethod":"mymethod.offloadingservice.com", "
       param1":"my 1st param","param2":"my 2nd parameter"}' \
7    https://172.28.48.31/offloadingservice/mymethod
8
9  // 1.2. Response to the offloading request
10 Status: 201 Created
11 Location: https://restapi.bigmobile.com/offloadingservice/mymethod
       /job_201412011859_0002
12
13 // 1.3. JSON formatted Body of the response to the offloading
       request
```

```
{
  "createdAt": "2014-12-01T18:59.00021Z",
  "objectId": "job_201412011859_0002"
}
```

6.4 Computation Offloading Service

Once an offloading request is received by the computation service in the front-end cloud service shown in 5.5, it is queued into the message queue to guarantee reliable message delivery within the BigMobile system. The BigMobile computation interface allows you to create and run Map/Reduce jobs and typical binary form of jobs with any executable or script as the mapper and/or the reducer. The interface achieves this by converting the offloading request through the RESTful API detailed in Section 6.3 into a form of streaming jobs, and submit it to Hadoop's YARN [71], responsible for job scheduling. The offloading service (compiled in OffloadingService.jar) to host the offloaded method from mobile client is previously given in Listing 3.3, will be hosted by cloud resource on-demand. The benefit of having the interface for the job submission is that application developers who do not have any knowledge of the map/reduce programming model can place a job onto computation resources in Hadoop as it takes any executable or script as a job.

Listing 6.3: An example of the job submission onto Hadoop's YARN where the BigMobile's computation interface converts the offloading request into a streaming form of job

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper "java -jar OffloadingService.jar $param1 $param2" \
    -file OffloadingService.jar  \
    -file https://172.28.48.31/offloadingservice/UserObject/
```

```
PbednuqPvsm
```

In Figure 6.2, the submitted offloading request via the RESTful APIs must be converted into a Hadoop understandable command. Listing 6.3 presents an automatically generated script, where the computation interface converts the offloading request (seen from Listing 6.2) into a streaming job that Hadoop's YARN can understand. There are options to specify input (line 2) and output directory (line 3). The request is mapped onto mappers with mypackage.jar on java runtime (line 4). The binary executable must be attached as in line 5. Two parameters $param1$ and $param2$ will be replaces with real values: "my first param", and "my second param", respective at runtime; In addition, one can attach pre-uploaded files in Listing 6.1 from the cloud storage to the job submission (line 6). Once the job is submitted, the provided url (line 8) in Listing 6.2 allows to track the status of the job.

Recall the mobile cloud application for automatic speech recognition described in Chapter 5. The computation offloading service roles as a batch processor for generating personalized acoustic models and language models for individual speakers. In 100 million (M) user scenario, the size of data can easily pile up to 6.2PB. For agile and better-personalized voice-enable services, we may need more frequent re-computation to keep up-to-date models. A typical approach in large data processing is the use of map/reduce programing model. We however note that none of speech libraries directly support the map/reduce, requiring lots of library modifications, which are obviously inefficient. The basic goal of the execution engine support is to have speech library run without any modification.

To this end, a Hadoop streaming interface meets our requirement. It is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. Some of usage examples are detailed in [71]. When an executable is specified for mappers, each mapper task will launch the executable as a separate process when the mapper is ini-

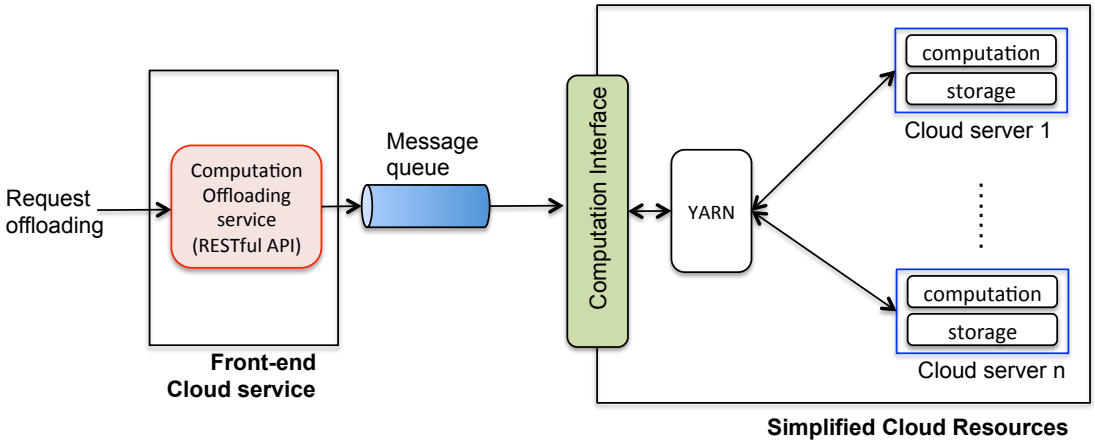

Figure 6.2: A computation offloading scenario that distributes the task into multiple computing resources (or slots) by YARN of Hadoop. The submitted offloading request via the RESTful APIs is converted into a Hadoop understandable command as shown in List 6.3.

tialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the stdin of the process. In the meantime, the mapper collects the line oriented outputs from the stdout of the process and converts each line into a key/value pair, which is collected as the output of the mapper. In our execution, we do not require reducers to execute. This leads to a potential problem of resource under-utilization in Hadoop v1 since pre-assigned mapper slots cannot be shared with reducer slots leaving them idle. In the worst case, 944 mappers can run when no reducers among 590 slots in our Hadoop v1 cluster are at work. This is a well-known problem extensively studied in the literature [75]. Fortunately, Hadoop v2 resolves this issue by centralizing resource management of mappers and reducers in one place.

Based on our implementation, we made several changes in the Hadoop Streaming interface (1.1.2.24 for Hadoop v1 and 2.2.0 for Hadoop v2) in following ways. (1) The speaker adaptation with Sphinx [65] requires to have output files not from streaming output as known as stdout. Although the interface supports -output option to specify an output directory, it is a place for streaming output, and all the intermediate files are lost eventually during the execution. (2) The current interface does not include an option for file compression including input and output. Based on the time saving we obtain in

our experiments, we suggest to have an option to specific whether or not input/output streaming or files are compressed before or after processing.

CHAPTER 7

Evaluation

We consider two set of MCC applications for evaluation of BigMobile: (1) A conventional type of computationally intensive applications: puzzle game, chess game, and cryptographic algorithm; (2) A new type of data-intensive application: the personalized speech recognition application, where the creation of personalized acoustic models requires the huge amount of both computation and data as detailed in Chapter 5. The metrics we consider is energy and time.

We note that the different wireless network connectivity has been well studied: *WiFi only* such as Spectra [20], Chrome [21], Cloudlets [15], MobiCloud [17], Hyrax [22], Scavenger [19]; *3G/4G-LTE only* such as GeoServ [3], VeSense [10] [11], mHealthMon [12] [13]; *Both WiFi and 3G/4G-LTE* found in MAUI [14], ThinkAir [18], ClondCloud [16]. We simplify our evaluation by making assumptions: only 3G/4G-LTE network is available; a mobile device is stationary. Every result is obtained by running the program 20 times for each scenario and averaged them at the end; there is a pause of more than 1 minute between two runs. The typical RTT of the 3G/4G-LTE network in our setting is around 100ms on average.

The rest of the chapter is organized as follows: two types of evaluation of BigMobile system: simple benchmark mobile applications (Section 7.1) and data-intensive mobile application (Section 7.2). The rest of chapter extends the performance evaluation to the cloud side where we present comparisons of Hadoop v1 and Hadoop v2, Hadoop overhead, optimization techniques, and small/large file support in Section 7.3. Finally the performance of the message queue is presented in Section 7.4.

7.1 Computation-intensive Application Benchmarks

There are two types of experiments considered in computation-intensive application benchmarks;

- The first type of applications such as chess and puzzle game are well studied in MAUI [14] and ThinkAir [18]. They require for small data transfer and benefit from offloading with respect to saving time more than energy;
- Second, we further study applications such as cryptographic algorithm DES and Huffman compression, which require relatively large data transmission than the first category.

We present the time saving followed by the energy saving. BigMobile framework minimizes energy consumption as well as execution time as the main optimization problem is formulated by the weighted tradeoffs between mobile execution and cloud execution with respect to energy costs and time cost as defined in Equation 4.1. We set the weight w to 0.5 for fairness in tradeoff. The overall procedure of offloading for evaluating applications follows the sequence explained earlier in Section 2.4.

7.1.1 Time Saving

Figure 7.1-7.4 plot the average runtime for each benchmark application, where a solid gray-coded bar presents the cost of computation and a diagonally patterned bar presents the cost of communication for data transmission from mobile to cloud. An offloading consistency factor for each application varies from 0 to 5. The value 0 denotes the local execution, and the value ranging from 1 to 5 denotes the degree of concurrent parallel offloading. For instance, BigMobile allows offloading up to three different methods at once if the offloading concurrency factor (OCF) is set to 3. As expected, the crypto-DES (Figure 7.3) and Huffman compression (Figure 7.4) require more time for transmitting data to cloud than the time to compute offloaded methods. On the other

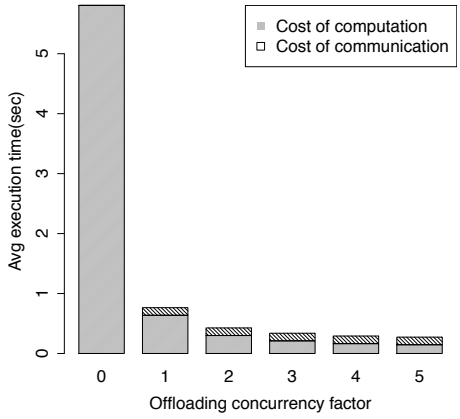

Figure 7.1: Execution time of chess game with 30 moves.

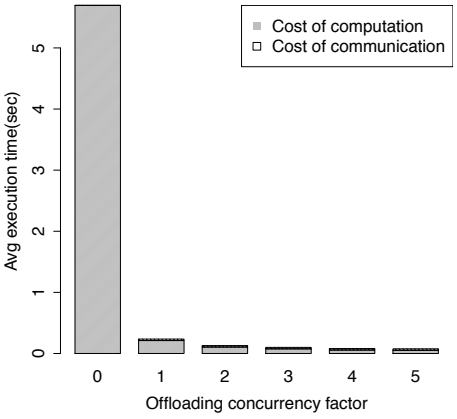

Figure 7.2: Execution time of puzzle game with 30 moves.

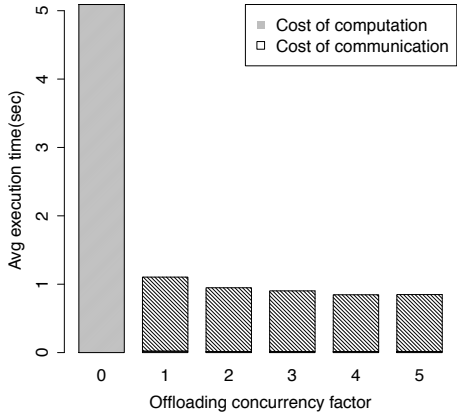

Figure 7.3: Execution time of cryptographic algorithm DES for 10 killobytes (KB) data encryption.

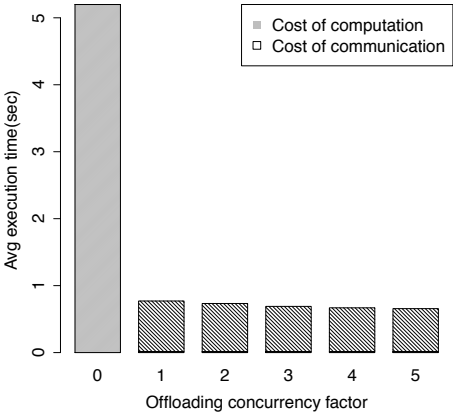

Figure 7.4: Execution time of Huffman algorithm for 10 killobytes (KB) data compression.

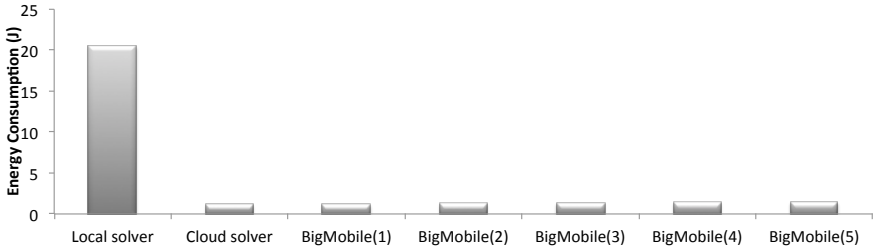

Figure 7.5: The energy consumed by the chess game application for different offload solvers.

hand, the chess (Figure 7.1) and puzzle (Figure 7.2) require less data transmission time due to the small size of data compared to the first category of applications. In summary, the relative offloading benefit compared to the local execution is 6x, 12x, 16x, 19x, 20x faster in the chess; 23x, 42x, 55x, 69x, 72x faster in the puzzle; 5x, 5x, 5x, 5x, 6x faster in the DES; 5x, 6x, 6x, 6x, 6x faster in the Huffman compression as OCF gets bigger from 1 to 5.

7.1.2 Energy Saving

The energy saving is a major concern for MCC applications although time saving is considerable. The corresponding energy consumption for each application is plotted from Figure 7.6 to Figure 7.9. The figures include the distribution of energy consumption in barplots, where 0%, 25%, 75%, 100% from bottom to up, and a mid bar for mean. In fact, the resultant energy saving through concurrent parallel offloading is significant compared to the local execution: compared to the local execution is 3x, 6x, 7x, 9x, 10x more energy-efficient in the chess; 12x, 22x, 30x, 37x, 39x more energy-efficient in the puzzle; 4x, 4x, 4x, 5x, 5x more energy-efficient in the DES; 3x, 3x, 3x, 3x, 4x more energy-efficient in the Huffman compression as the offloading concurrency factor varies from 1 to 5.

Table 7.1 presents the summary of energy saving and time saving for group 1 (chess, puzzle) and group 2 (crypto-DES, Huffman) applications. For instance, 3x refers to three times faster or more energy-efficient than local execution when offloading. The range of values varies when the concurrent and parallel offloading factor increases.

7.1.3 Energy Consumption for BigMobile Solver

We compare the energy required for the BigMobile solver with other solvers such as MAUI, ThinkAir, and CloneCloud. As shown in MAUI, we implement the MAUI solver (corresponding to single cloud-based ILP solver) as a baseline to compare en-

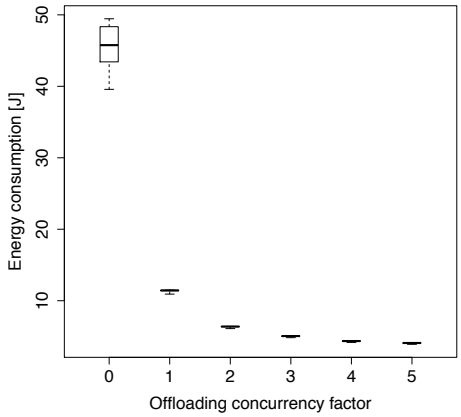

Figure 7.6: Energy consumption of chess game with 30 moves.

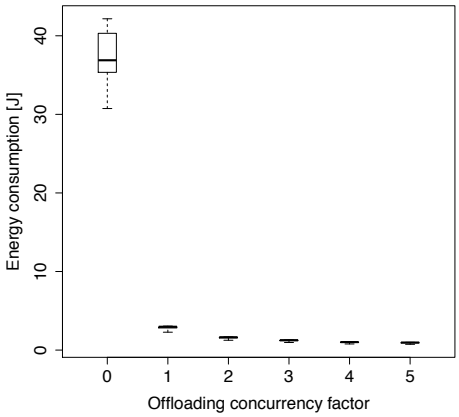

Figure 7.7: Energy consumption of puzzle game with 30 moves.

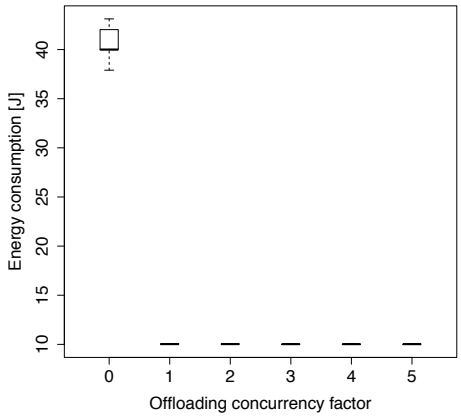

Figure 7.8: Energy consumption of cryptographic algorithm DES for killobytes (KB) data encryption.

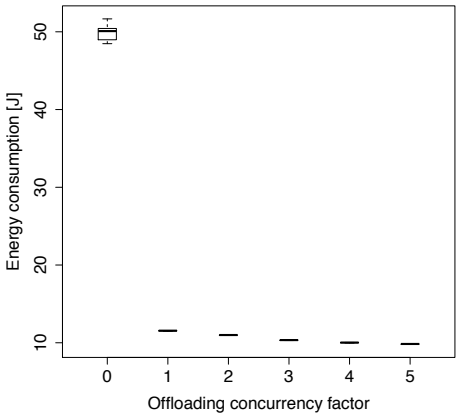

Figure 7.9: Energy consumption of Huffman algorithm for 10 killobytes (KB) data compression.

Apps	Energy Saving	Time Saving
Chess	3x - 10x	6x - 20x
Puzzle	12x - 39x	23x - 72x
Crypto-DES	4x - 5x	5x - 6x
Huffman	3x - 4x	5x - 6x

Table 7.1: Summary of energy saving and time saving for group 1 (chess, puzzle) and group 2 (crypto-DES, Huffman) applications.

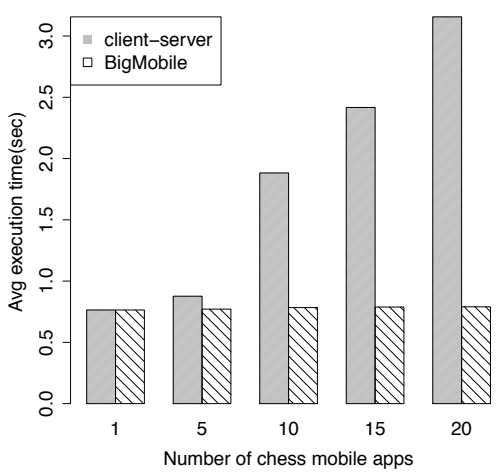

Figure 7.10: The comparison of offloaded execution time for the chess game application.

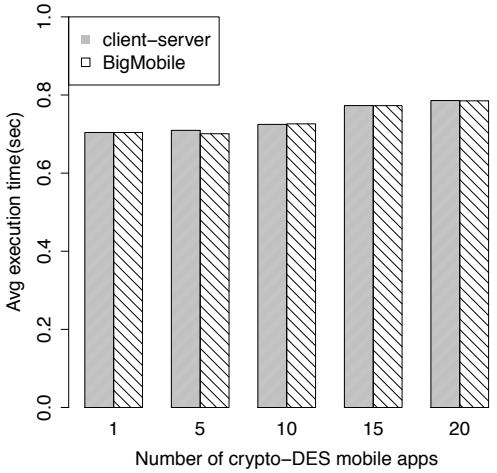

Figure 7.11: The comparison of offloaded execution time for the cryptographic algorithm (DES) application.

ergy consumption with BigMobile. The baseline implementation constitutes a local solver and cloud solver. Unlike them, the cloud version of BigMobile takes multiple clouds as resources for offloading computation, where there exists more profiling cost than competitors as described in Section 4.4. In Figure 7.5, we presents aforementioned schemes' energy consumption for a simple chess game. The number from 1 to 5 coming after the *BigMobile* represents the number of multiple clouds we solve as the integer programming problem (IP) formulated in Equation 4.1. It is obvious that the local solver consumes much more energy than the cloud version. The energy for the BigMobile with a single cloud is compatible for the cloud solver. Moreover, the overall energy consumption for the BigMobile with the consideration of multiple clouds settings stays acceptable from a practical viewpoint. This means that the BigMobile solver performs reasonably well w.r.t. energy as the number of partitions modeled in the IP problem grows due to the multiple clouds consideration.

7.1.4 Multiple Clouds Support

We study the scalability of the system as the number of offloading requests increases. A typical client-server approach is to have a single cloud server to support all the incoming requests. We compare BigMobile with the client-server scheme. The client-server represents a serial offloading to a single cloud server, and most of the systems discussed in Section 1.1 except for ThinkAir [18] falls in this category. BigMobile however supports multiple servers to be scaled up to the maximum number of slots in Hadoop, typically several thousand slots for several hundreds machines. We present two type of representative applications: chess game that requires small data transmission when offloading; and cryptographic algorithm DES that requires relatively large data transfer to the cloud. In chess, BigMobile outperforms as the number of mobile applications gets bigger in Figure 7.10. The main reason is that the cost of communication stays similar when the cost of computation is significantly reduced due to multiple servers added in BigMobile. On the other hand, the crypto-DES shows different outcomes since the overall cost of communication is too high compared to that of computation (Figure 7.11). Thus, no benefit is observed even when more cloud servers are provisioned in BigMobile. We did not present the analysis of energy consumption comparison between the two schemes since they are almost same by exploiting parallel offloading. Therefore, we observe that BigMobile's scalable cloud support is more suitable for mobile applications that do not require lots of data transfer to cloud when offloading. Rather, BigMobile applications can benefit from fetching data from the cloud storage.

7.2 Data-intensive Application Benchmark: Automatic Speech Recognition

We extend the evaluation of BigMobile to more complex mobile applications, which are an automatic speech recognition and its energy-hungry computation on personalized acoustic model (PAM) generation. The PAM task consists of six different jobs as

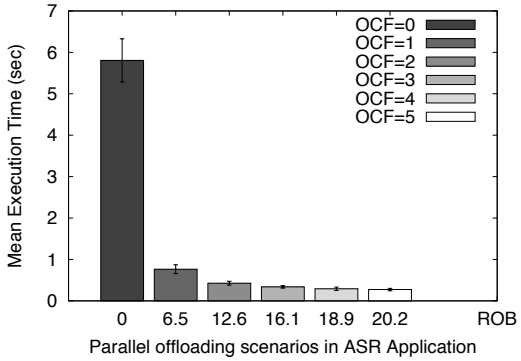

Figure 7.12: The comparison of execution time for the ASR mobile application.

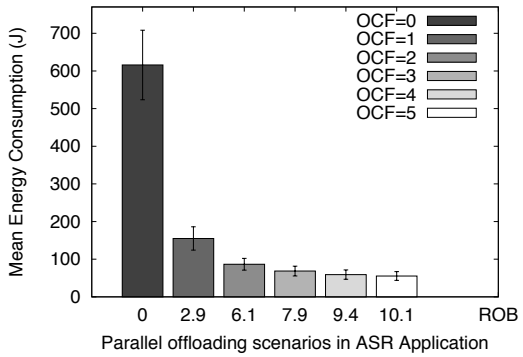

Figure 7.13: The comparison of energy consumption for the ASR mobile application.

described in Section 5.2 using the Sphinx library [65]. A major discrepancy between the ASR app and previously evaluated benchmarks apps in Section 7.1 is the degree of big data usage from the cloud storage. The ASR app we consider utilizes more than 6 petabytes of associated files from the cloud when executing offloaded computation in the scenario of 100 million ASR users. We first evaluate the energy saving in the ASR app on top of BigMobile in this section, and go on to evaluate the performance of cloud infrastructure of BigMobile in Section 7.3. Figure 7.13 presents the comparison of execution time for the ASR app presented in Chapter 5. BigMobile solves the optimization problem to minimize both energy and time through parallel and consistent offloading to multiple clouds. The energy consumption decreases as the offloading consistency factor (OCF) increases from 1 to 5. The value 0 refers to the local execution. The relative offloading benefit is presented below to each bar to present the degree of benefits from offloading, compared to the local execution (OCF=1). The figure shows the parallel and consistent offloading can save energy up to 10.1 times. As an additional benefit of offloading, we present time saving in Figure 7.12, where offloading enables for the ASR app to execute up to 20.2 times faster than the local execution.

7.3 Performance of Data-Intensive Computational Cloud

We evaluate multiple clouds-based BigMobile system with two different configuration of Hadoop clusters: Hadoop v1 and Hadoop v2. This section focuses on the performance and scalability of BigMobile's data-intensive clouds rather than energy consumption in mobile devices to effectively support up to 100 millions apps. Hadoop v1 cluster has the capability of 944 slots for mappers and 590 slots for reducers. Both clusters hold homogeneous machines with Intel Xeon 2.00GHz 12 cores, 64GB memory. The later section describes four different optimization techniques to enhance the performance in computation assuming that the speaker adaptation capability is bounded by a specific speech library, in this case CMU Sphinx [65]. This optimization goal is reasonable since we aim to build a scalable voice-enabled service by the help of big data platform.

We briefly describe experiment settings for the personalized acoustic models (PAMs) generation jobs on Hadoop. Figure 5.6 presents a whole flow from a job submission client to the compute node that performs actual Sphinx jobs [65]. A one PAM creation requires to call six different Sphinx programs (each program corresponds to a job) in three different speech libraries: SphinxBase, PocketSphinx, and SphinxTrain. In the optimization discussion, we perform a workflow compaction from six to one job to reduce the total execution time.

Our description follows the order of numbers we put in the figure. Once the job client requests a workflow execution via the BigMobile computation interface (1), the interface sends this request to a Hadoop job manager called YARN (2). Note that Hadoop v1 calls it a job tracker while Hadoop v2 calls it an application manager. The job manager assigns empty slots for map jobs to each of requested Sphinx jobs (3). After the job assignment, the rest of unassigned jobs are left in the job queue to wait for other empty slots. The job manager is also responsible for shipping all the required program binaries, specified files and directories to a temporary directory on the assigned

compute node. Note that this shipping can be a bottleneck if large files are transferred. Usually Hadoop streaming supports a distributed cache up to 10GB to avoid file system access, leading to the performance degrade. One challenge is the large number of PAMs generation requires PB scale file read from somewhere. It is obvious that the 10GB distributed cache cannot fit to our scenario of execution. This situation leads for each compute node to read required 62MB uncompressed (4-1) or 35MB compressed files (4-2) from HDFS. Another options are reading files from object storage (4-3), data storage (4-4), or external distributed file systems such as Google object storage (4-5). Such five considerations are motivated for performance improvement that we will deeply discuss in the later section. Finally, output files including a PAM (30MB uncompressed, 17MB compressed) needs to be shipped to user-specified file systems (can be any of five options aforementioned) after the job completion. The last mission of the computing node also incurs the large write file operation time.

7.3.1 Result and Optimization

We run the proposed system in two different clusters: 59 nodes running Hadoop v1 cluster and 15 nodes running Hadoop v2 cluster. Hadoop v1 cluster has the capability of 944 slots for mappers and 590 slots for reducers. Both clusters hold homogeneous machines with Intel Xeon 2.00GHz 12 cores, 64GB memory. The later section describes four different optimization techniques to enhance the performance in computation assuming that the speaker adaptation capability is bounded by a specific speech library, in this case CMU Sphinx [65]. This optimization goal is reasonable since we aim to build a scalable voice-enabled service by the help of big data platform.

PAM Computation Metric. Terasort is a well-known tool for measuring the sorting efficiency of the Hadoop cluster based on 1 terabytes of data [76]. The sorting efficiency is defined in $\frac{MB}{(sec)*(core)}$. Inspired by the metric of terasort, we define the efficiency of PAM computation metric E as

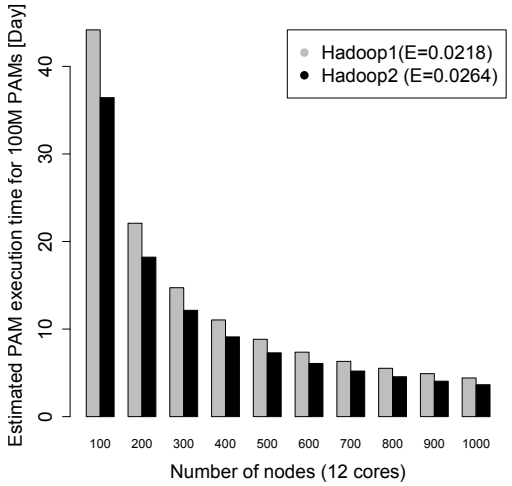

Figure 7.14: The estimated execution time for 100 millions(M) PAMs generation in hour on both Hadoop v1 and Hadoop v2 based on Eq. (7.1).

$$E = \frac{P}{T * C}, \tag{7.1}$$

where P denotes the total number of PAMs generated, T denotes the total elapsed time to generate PAMs, and C denotes the total number of CPU cores used in the PAM generation. This metric is useful to compare one Hadoop cluster with the other, yet the different number of cores.

7.3.2 Hadoop v1 cluster vs. Hadoop v2 cluster

As a preliminary execution, we run 40K PAMs jobs on a single machine-based Hadoop v1 cluster, and it took 7.8 days to complete. As a parallelism test, we also run the same jobs under the condition that 25 job submission clients concurrently submit each job to the 59 nodes Hadoop v1 cluster, and this run took 3 days to complete. In the later experiment, the concurrent number of mappers does not go up 30 ,meaning that parallelism is not fully taken into account. Given that amount of computation, we investigate the performance of both Hadoop v1 and Hadoop v2 cluster, with no optimization technique applied. We study the overall PAM generation time as the number of PAMs grows on

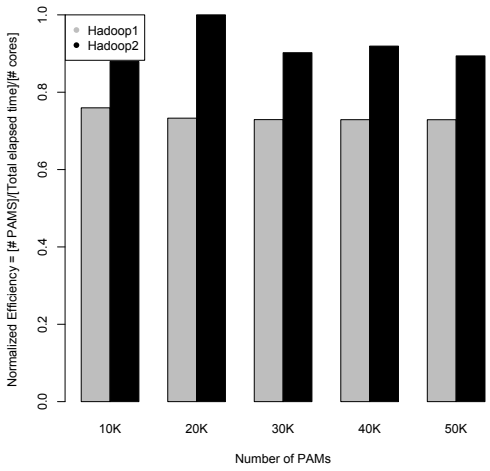

Figure 7.15: PAM efficiency comparisons between Hadoop v1 and Hadoop v2 based on the metric E defined in Eq. (7.1).

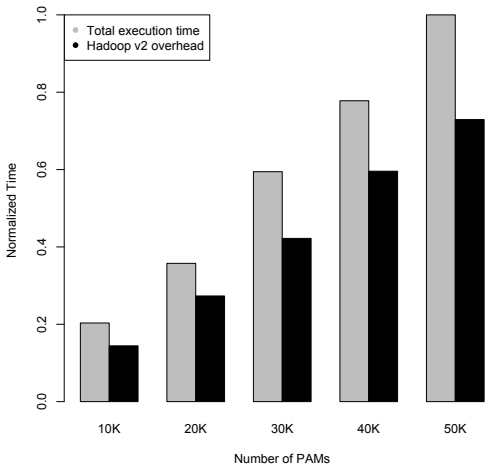

Figure 7.16: The measurement of normalized Hadoop overhead including package shipment to compute node, file operations.

both clusters. For the 40K PAMs scenario, Hadoop v1 finishes jobs in 1.25 hours and Hadoop v2 does them in 2.25 hours Note that Hadoop v1 cluster has 3 times more nodes than Hadoop v2 cluster. Given that node difference, such result is not shocking. Figure 7.15 provides an effective way to compare the efficiency metric defined in Eq. 7.1. For given scenarios from 10 thousands (K) to 50 thousands (K) PAMs, Hadoop v2 is more efficient about 15% to 25% depending on scenarios.

The reason why we stopped our experiments on 80K for Hadoop v1 and 50K for Hadoop v2 is that we got a Hadoop history server crash over that number. We find that excessive syslog (10s of GB) from Sphinx library causes an out-of heap memory problem although we run stable versions of Hadoop v1 and Hadoop v2.

PAM Efficiency Metric. As we define an efficiency metric in Eq. 7.1, we here verify how good the metric provides estimated time for the given number of PAMs to be generated in both Hadoop v1 (H1) and Hadoop v2 (H2). E_{H1} and E_{H2} denote the efficiency of PAM computation for H11 and H2, respectively. E_{H1} and E_{H2} are calculated at 80K and 50K PAMs scenarios, respectively. The x-axis presents the number of PAMs under

either H1 or H2. Except for the case of H2.20K, all the other cases sit within 95% of confidence interval. This verifies the effectiveness of the metric Eq. 7.1 experimentally.

The following experiment in Figure 7.14 goes on to say how both clusters can be scalable as the number of PAMs grows, and how many nodes are requires to meet the real world service scenario. For instance, when we may need to update 100 millions(M) PAMs every week, the number of nodes requires to meet the requirement is around 700 nodes for Hadoop v1 and 600 nodes for Hadoop v2. This estimation is very important to consider when designing the scalable voice-enabled system for commercialization. The trend in the estimated time stays in log scale as we put more nodes in clusters. In a cost effective aspect, 700 nodes and 1000 nodes do not make much difference in performance for one week cycle in PAM update. Given that typical price of 4K-5K dollars per machine, 300 machines can affect a huge memory loss. The PAM update period is one another factor to consider in this direction. If we set the update period as 2 weeks, 400 and 300 nodes are enough for Hadoop v1 and Hadoop v2 respectively, saying a half of investment on cluster purchase. It estimates 100M PAMs generation time in 4.4 days on 1000 nodes H1 and H2 cluster. Also both H1 and H2 can finish 1M PAMs generation within one day on the same hardware settings.

7.3.3 Hadoop Overhead

As an optimization effort, we study the overhead caused by Hadoop including job coordination, file operations, cluster management overhead, etc. Figure 7.16 compares the total job execution time for 10K to 50K PAMs scenarios with the Hadoop overhead in time. A way we measure the overhead is that we eliminate all the Sphinx programs while keeping all the other operations including file read and write. We obtain around 25% of time difference between the two if we normalize the figure. In other words, Sphinx computation running on Hadoop only takes 25% of the total execution time, and this gives us to consider more optimization opportunities such as file compression, workflow compaction. In fact, Sphinx requires 6 different jobs to run, and we optimize

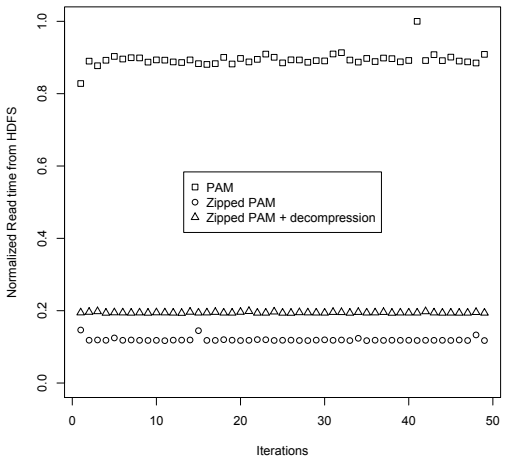

Figure 7.17: The effectiveness of file compression in HDFS read operation.

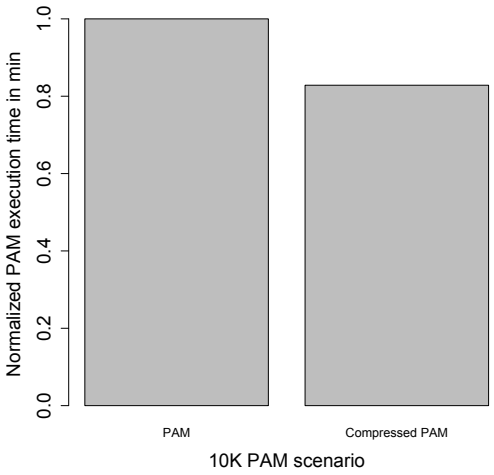

Figure 7.18: The effectiveness of file compression in HDFS read operation becomes real benefit in the total PAM execution time in 10K PAMs scenario.

6 jobs to 1 job for each PAM running on a Hadoop compute node. By such workflow compaction, we save 62.5% of the total execution time since this save is mainly due to the Hadoop overhead.

7.3.4 File Compression

A file compression technique is another interesting area to explore for the execution time optimization. Each PAM generation requires 62MB of files shipping to a temporal directory on an assigned Hadoop compute node. The file compression reduces it to 35MB meaning that we can save a half of file reading time. We however need to consider decompression time for the compressed file after downloading. Figure 7.17 consider such scenario to study the feasibility of file compression benefit in time. We set up a script to read 50 iterations of PAM file from HDFS and zipped (=compressed) PAMs to compare with the former. The bottom dots present reading time for compressed PAM while the baseline for uncompressed PAM reading time is on the top dots. After considering the decompression time after downloading we obtain dots in

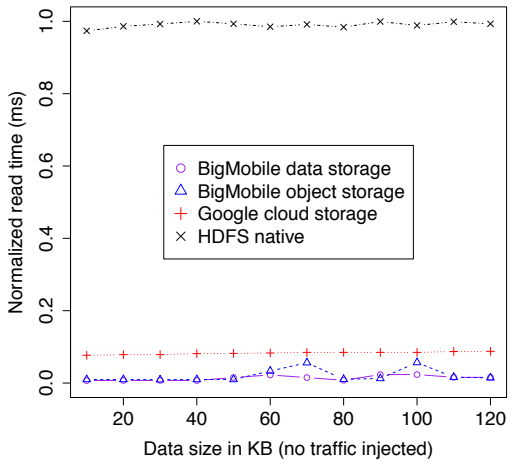

Figure 7.19: The read performance comparisons for small file support.

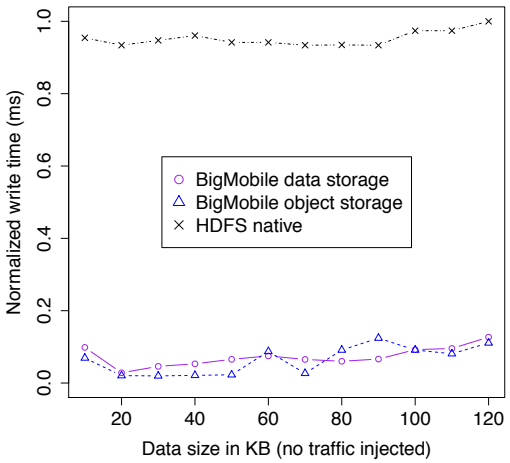

Figure 7.20: The write performance comparisons for small file support.

the middle that has 70% of file reading time saving compared to the baseline. This means HDFS file operation has lots of overhead in time, and file compression may effectively reduce the operation time. To prove such consideration we set up 10K PAM execution on both uncompressed PAM and compressed PAM scenario in Figure 7.18. If we normalize two bars in the figure, we obtain 18% of difference in between the two. This proves the effectiveness of the file compression.

7.3.5 Small File Support

Other than PAM generation, the ASR creates lots of small files including speech files (1KB-100KB typically) and text logs (1KB-50KB). To support such transactions for file write and read operations, we compare various existing options we can consider. Figure 7.19 presents the measure of file read time ranging from 10KB to 120KB considering four different technologies available: the method starting with BigMobile denotes developed systems. In fact, most of developed systems outperform well-known Google object storage. Among BigMobile systems, data storage and object storage are the one to pursue. The data storage is a key-value based system that stores (key, value) pairs

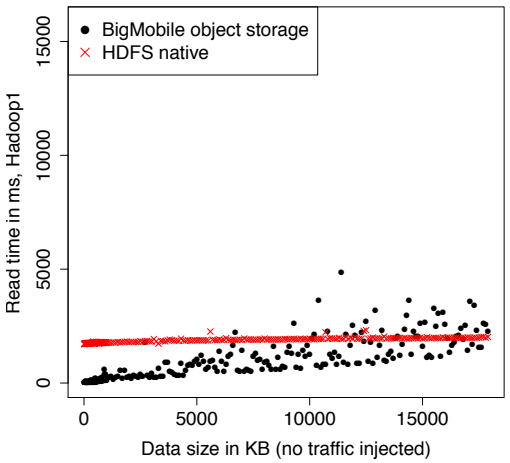

Figure 7.21: The read performance comparisons for the large file support under no traffic injected.

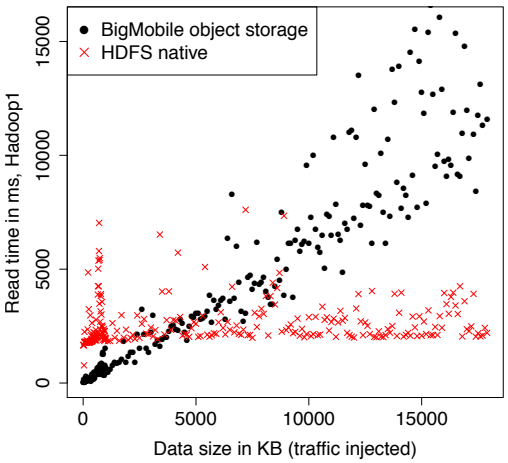

Figure 7.22: The read performance comparisons for the large file support under very large traffic injected.

in tables while the object storage is a hybrid file system that utilizes HBase and HDFS in one place to make it better performed for various file size. Figure 7.20 presents write time of BigMobile systems. Although the number of read is much higher than the number of write in a typical ASR system, we want to make sure that system does not be overkilled by the write operation. It seems both the data storage and object storage performs fairly good enough to consider within the small file size range.

7.3.6 Large File Support

As aforementioned, large files up to 30MB must be effectively supported for the PAM generation. We observe that the data storage needs to be careful when considering due to the nature of database system, where large files are less likely to be effectively supported in terms of fast read and write. The rest of options we have are that HDFS native APIs and BigMobile object storage. The measurement is performed in the early morning of weekdays to make sure that the cluster does not have other large file operations. We only present the performance of reads since the number of reads is much larger

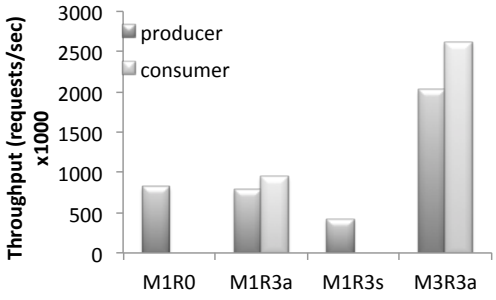

Figure 7.23: The performance of BigMobile's message queue in offloading requests per second for both consumer and producer.

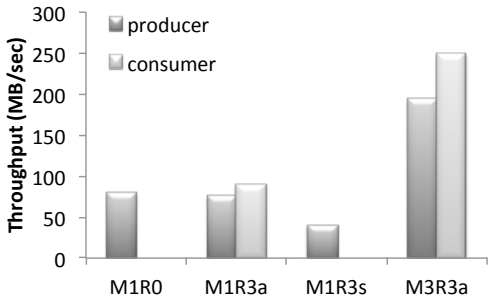

Figure 7.24: The performance of BigMobile's message queue in throughput (MB/sec) for both consumer and producer.

than the number writes for the large files. Figure 7.21 presents read time for various file size under no traffic injected to H1 cluster, and the object storage outperforms over the HDFS native in most of file sizes. In order to observe the situation, where lots of file operation traffic injected, we generate 944 mappers, each of which writes 30MB file to HDFS and reads it five times accordingly. We consider this dummy traffic as very large traffic since the number of mappers is bounded by 944 in Hadoop v1 cluster. We measure read time of various files with different size while the underlying file operations exist. In Figure 7.22, we observe the shift of the crossing point (15MB to 5MB) compared to Figure 7.21. This means that the BigMobile object storage can provide better performance under 5MB of files for given large traffic. For the given workload in the cluster, it is recommended to adapt the file system in a hybrid way, where the HDFS native is preferred over 5MB file size.

7.4 Performance of Message Queue

The reliable message queue implemented in BigMobile is to target high performance throughput from producers to consumers through brokers. We set up six commodity machines, where each node comprises with CPU(2x Intel Xeon E5-2620v2@2.1GHz), 15M Cache 7.2GT/s QPI, RAM (4x 16GB 1600 MT/s), HDD(8x 3TB 7.2k RPM Near-

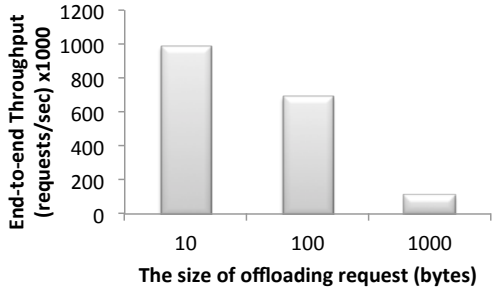

Figure 7.25: The end-to-end performance of BigMobile's message queue in offloading requests per second from consumer to producer through brokers.

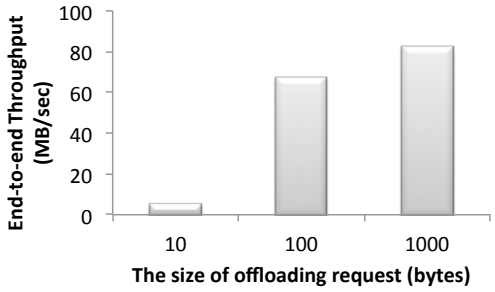

Figure 7.26: The end-to-end performance of BigMobile's message queue in throughput (MB/sec) from consumer to producer through brokers.

Line SAS 6Gbps), NIC (Broadcom 57800 2x10Gb DA/SFP+2x1Gb BT). We test a replication factor up to three as we need. The message queue allows us to add more partitions and spread data onto more machines to scale our cluster horizontally by design.

As a workload, we create a offloading request and then produce 50 million small (100 byte each) requests as quickly as possible from a single thread. A set of performance measurement for producers is following: (1) single producer with no replication (M1R1); (2) single producer with 3 synchronous replications (M1R3s); (3) single producer with 3 a-synchronous replications (M1R3a); 3 producers with 3 a-synchronous replications (M3R3a). Similarly, a set of experiments for consumers is follows: (1) single consumer with 3 a-synchronous replications (M1R3a); (2) 3 consumers with 3 a-synchronous replications (M3R3a). Figure 7.23 presents throughput of the message queue in terms of offloading requests per second while Figure 7.24 presents its throughput in terms of Mega bytes per seconds.

Producer Throughput. For a single producer scenario in Figure 7.23, the producer with no replication (M1R0) is 4% outperformed than M1R3a and 94% outperformed than M1R3s since the 3x replication is costly. M1R3a's throughput is 86% better performed than MaR3s due to the synchronousness property. With the three producers scenario, the producers (M3R3a) with asynchronous replication can process 2.4 times

more requests per second than M1R3a. Figure 7.24 presents very similar trends in throughput in MB/sec.

Consumer Throughput. We study throughput of consumers in two scenarios: single consumer (M1R3a) and 3 consumers (M3R3a) for 3 asynchronous replications. M3R3a outperforms 2.7 times better than M3R3a in both requests/sec (Figure 7.23) and MB/sec (Figure 7.24).

Overall Throughput of Message Queue. The overall throughput of the message queue in BigMobile is studied by varying the size of offloading requests from 10 bytes to 1000 bytes. For this test we run one producer and one consumer on a six partition 3x replicated topic that begins empty. The end-to-end throughput is upper-bounded by the producer throughput. Figure 7.25 presents the end-to-end throughput in offloading requests per second. The overall throughput decreases from 983,237 to 687,125 to 104,299 request/sec as the data size of message increases. This implies keeping the message size small is important in high performance message delivery. From the data transmission perspective, we can see that with the 10 byte messages we are actually CPU bound by just acquiring the lock and enquiring the message for sendingwe are not able to actually max out the network. However, starting with 100 bytes, we are actually seeing network saturation though the MB/sec continues to increase as our fixed-size bookkeeping bytes become an increasingly small percentage of the total bytes sent. In addition, Figure 7.26 presents the end-to-end throughput in the unit of MB/sec.

CHAPTER 8

Conclusion

We propose the data-intensive mobile cloud computing platform called BigMobile. First, the proposed system exploits a function-level parallel offloading method in a client-server paradigm with no assumption of offloading tasks to be preinstalled and no VM instance instantiation required. Second, BigMobile supports a set of interfaces for the fuse with ever-successful big data platform Hadoop. Fault-tolerance under unstable wireless medium is achieved by implementing the asynchronous offloading mechanism. The big data integration provides two benefits: integration of job scheduler and native access to distributed file system and possibly no SQL database. Third, the message queue achieves reliable and scalable offloading message delivery. Fourth, a BigMobile solver utilizes data collected by a profiler as input, and solves a multi-cloud partitioning problem that determines which remoteable methods should execute locally or remotely. The goal of the solver is to optimize the tradeoffs between mobile execution and cloud execution saving for a given mobile application.

In our evaluation, we present the performance using two types of mobile applications: the first type is a set of computation-intensive benchmarks, where energy saving is up to 39x and time saving is up to 72x by parallel offloading; the second is both data- and computationally-intensive mobile application - automatic speech recognition. We first verify the feasibility of speaker adaptation based on 107 testers' recordings and obtained improved recognition accuracy by 10%, and energy saving is up to 10x, followed by 20x of time saving by parallel offloading.

We also evaluate the performance of the BigMobile cloud. We report Hadoop v2

is 15%-25% more efficient than Hadoop v1 for given PAM scenarios, and Hadoop overhead is about 75% of the total execution time. With the offloading workflow compaction, we saved 65.2% of the total execution time, and file compression also contributed to save 70% of file operation time and 18% of the total PAM offloading time. For the best file system support, we explore several options: most of small files less than 120KB can be effectively supported by either the cloud object storage and the cloud data storage; large files less than 30MB can be effectively supported in a hybrid approach. Based on the PAM offloading computation metric with 95% of confidence interval, we obtain the estimated time: less than one day for one millions PAMs offloading and less than three days for 100M PAMs offloading under 1000 nodes cluster in Hadoop v2. The comprehensive evaluation of mobile and cloud in BigMobile concludes this thesis.

CHAPTER 9

Future Work

Platform issue. Platform Issue. Most of work in MCC has focused on Android OS-based program partitioning mechanisms so far. Recently, Tizen operating system (OS) has obtained growing attention as a smart home [52] [53] and IoTs platform [54], [60], [61], [87], [89], [90], [92], [93]. It an open and flexible operating system built from the ground up to address the needs of all stakeholders of the mobile and connected device ecosystem, including device manufacturers, mobile operators, application developers and independent software vendors. We plan to develop a full set of functionalities of BigMobile in Tizen to support IoT devices other than smartphones. We aim to run BigMobile applications on smartphones, tablets, netbooks, and in-vehicle infotainment system, and smart TV. The features of the target devices can be found in [66].

Security. In spite of the considerable efforts in MCC research community, there are a number of loopholes and challenges that still exist in the security policies of mobile cloud computing [55] [88] [91]. The survey critically investigates different security frameworks proposed for the MCC environment. Most of the discussed security frameworks offload processor intensive jobs on cloud due to the resource limitation of mobile devices. Although the offloading increases the processing capability of the mobile device, the mobile user has to pay while using the cloud resources in a pay-as-you- go manner. Most of the discussed security frameworks overlooked the tradeoff between the energy consumption on the device and the expense of using cloud resources while designing a security framework. A combination of cloud computing with virtualization provides efficient utilization of cloud resources with minimum cost. Despite the

advantages provided by virtualization in MCC, new security threats need to be tackled due to the lack of perfect isolation among various virtual machine instances running on the same physical server. The most challenging aspects in MCC are guaranteeing user privacy and the provision of mobile application security that uses cloud resources. To provide a secure MCC environment, service providers need to address issues pertaining to data security, net- work security, data locality, data integrity, web application security, data segregation, data access, authentication, authorization, data confidentiality, data breach issues, and various other factors. To achieve a secure MCC environment, security threats need to be studied and addressed accordingly.

Context Awareness We can see a big wave the Internet of Things (IoT) and IoT can be benefitted from lots of MCC properties in terms of computation offloading from weak sensor devices to rich devices including the cloud. Towards the Internet of Things (IoT), the number of sensors deployed around the world is growing at a rapid pace. These sensors continuously generate enormous amounts of data. However, in order to add value to raw sensor data we need to understand it. Collection, modeling, reasoning, and distribution of context in relation to sensor data plays critical role in this challenge. Context-aware computing has proven to be successful in understanding sensor data. [59] provides a comprehensive surveys in context awareness from an IoT perspective: It presents the necessary background by introducing the IoT paradigm and context-aware fundamentals. [58],[57], and [56] survey sensor-based activity monitoring, modeling, and recognition from which strengths and weaknesses of those approaches are highlighted which can provide intelligence for MCC as well. We believe the context awareness can improve the quality of service in MCC to better understand ambient settings.

REFERENCES

[1] eMarketer. (2014, January 16). *Smartphone Users Worldwide Will Total 1.75 Billion in 2014* [Online]. Available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

[2] S. Gunelius. (2014, May 3). *Mobile Devices to Surpass the Number of People on Earth* [Online]. Available: http://newstex.com/2014/05/03/mobile-devices-to-surpass-the-number-of-people-on-earth-infographic

[3] J. H. Ahnn, U. Lee, and H. J. Moon, "GeoServ: A Distributed Urban Sensing Platform," *CCGRID*, pp. 164-173, 2011.

[4] M. Satyanarayanan, "Fundamental challenges in mobile computing," *PODC*, pp. 17, 1996.

[5] J. Rybicki, B. Scheuermann, W. Kiess, C. Lochert, P. Fallahi, and M. Mauve, "Challenge: Peers on Wheels - A Road to New Traffic Information Systems," *MobiCom*, pp. 215-221, 2007.

[6] J. Rybicki, B. Scheuermann, M. Koegel, and M. Mauve, "PeerTIS: a Peer-to-Peer Traffic Information System," *VANET*, pp. 23-32, 2009.

[7] D. Kessens and T. Savolainen, "3G and IPv6 impact on battery life," *French IPv6 Worldwide Summit*, 2006.

[8] S. Liu, W. Jiang, and J. Li, "Architecture and Performance Evaluation for P2P Application in 3G Mobile Cellular Systems," *WiCom*, pp. 914 - 917, 2007.

[9] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: Video Streaming with WWAN Striping," *Multimedia*, 2006.

[10] J. H. Ahnn and M. Potkonjak, "VeSense: Energy-Efficient Vehicular Sensing," *IEEE 77th Vehicular Technology Conference*, 2013.

[11] J. H. Ahnn and M. Potkonjak, "VeSense: High-Performance and Energy-Efficient Vehicular Sensing Platform," *Journal of Pervasive and Mobile Computing*, vol. 12, pp. 112122, 2013.

[12] J. H. Ahnn and M. Potkonjak, "Toward Energy-Efficient and Distributed Mobile Health Monitoring Using Parallel Offloading," *IEEE 35th Engineering in Medicine and Biology Society*, 2013.

[13] J. H. Ahnn and M. Potkonjak, "mHealthMon: Toward Energy-Efficient and Distributed Mobile Health Monitoring Using Parallel Offloading," In *IEEE Journal of Medical Systems*, 37(5):1-11, Oct. 2013.

[14] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," *Proc. 8th ACM international conference on Mobile systems, applications, and services*, pp. 4962, 2010.

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

[16] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," *EuroSys*, pp. 301314, 2011.

[17] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: building secure cloud framework for mobile computing and communication," *SOSE*, pp. 2734, 2010.

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

[19] R. Iyer, S. Srinivasan, O. Tickoo, Z. Fang, R. Illikkal, S. Zhang, V. Chadha, P. Stillwell, and S. Lee, "Cogniserve: Heterogeneous server architecture for large-scale recognition," *IEEE Micro*, 31(3):2031, 2011.

[20] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 217226, 2002.

[21] R. Balan, M. Satyanarayanan, S. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," *Proceedings of the 1st ACM International Conference on Mobile Systems, Applications and Services*, pp. 273286, 2003.

[22] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using MapReduce," *Masters Thesis, Carnegie Mellon University*, 2009.

[23] M. Shirax, A. Gani, R. H. Khokhar, and R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," *IEEE Communications Surveys and Tutorials*, 15(3):1294-1313, 2012.

[24] L. Rabiner and B-H Juang, "Fundamentals of Speech Recognition," *Prentice-Hall International, Inc.*, 1993.

[25] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile Cloud Computing: A survey," *Journal of Future Generation Computer Systems*, 29:84-106, 2013.

[26] A u. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A Survey of Mobile Cloud Computing Application Models," *IEEE Communications Surveys and Tutorials*, 16(1):393-413, 2014.

[27] E. Tse, Z. Luo, and N. Yigitbasi. (2014, Oct. 7). *Using Presto in our Big Data Platform on AWS* [Online]. Available: http://techblog.netflix.com/2014/10/using-presto-in-our-big-data-platform.html

[28] P. Vagata and K. Wilfong. (2014, Apr. 10). *Scaling the Facebook data warehouse to 300 PB* [Online]. Available: https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb

[29] MIT Kerberos. (2014, Oct. 16). *Kerberos: The Network Authentication Protocol* [Online]. Available: http://web.mit.edu/kerberos/.

[30] P. Ganore. (2014, Feb. 26). *Google scans more than 20 petabytes of data per day* [Online]. Available: https://esdsdatacenter.wordpress.com/2014/02/26/google-scans-more-than-20-petabytes-of-data-per-day

[31] M. Asay. (2014. Sep. 12). *Why the world's largest Hadoop installation may soon become the norm* [Online]. Available: http://www.techrepublic.com/article/why-the-worlds-largest-hadoop-installation-may-soon-become-the-norm

[32] Jong Hoon Ahnn, "Toward Personalized and Scalable Voice-Enabled Service Powered by Big Data," *IEEE International Conference on Big Data*, 2014.

[33] Jong Hoon Ahnn, "Scalable Big Data Computing for the Personalization of Machine Learned Models and its Application to Automatic Speech Recognition Service," *IEEE Workshop on Scalable Machine Learning: Theory and Applications*, 2014.

[34] Jong Hoon Ahnn, "Big Data Computing for the Personalization of Services and its Application to Automatic Speech Recognition," *IEEE/ACM Symposium on Big Data Computing*, 2014.

[35] *Android NDK* [Online]. Available: https://developer.android.com/tools/sdk/ndk

[36] *Soot- A framework for analyzing and transforming Java and Android Applications* [Online]. Available: http://sable.github.io/soot

[37] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, 44(4):585-591, 1997.

[38] *Power Monitor - Monsoon Solutions* [Online]. Available: https://www.msoon.com.

[39] (2013, Nov. 23). *Caiso source user guide* [Online]. Available: http://www.caiso.com/1ca6/1ca67a5816ee0.pdf

[40] NYISO Auxiliary Market Operations. (2013, Oct. 3). *NYISO Emergency demand response program manual* [Online]. Available: http://www.nyiso.com

[41] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick. Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," *CODES+ISSS*, pp. 105-114, 2010.

[42] S. Aman, Y. Simmhan, and V. K. Prasanna, "Improving energy use forecast for campus micro-grids using indirect indicators," *IEEE Workshop on Domain Driven Data Mining*, 2011.

[43] B.-J. Chen, M.-W. Chang, and C.-J. Lin, "Load forecasting using Support Vector Machines: a study on EUNITE competition 2001," *IEEE Transactions on Power Systems*, 19(4):18211830, 2004.

[44] H. Mori and A. Takahashi, "Hybrid intelligent method of relevant vector machine and regression tree for probabilistic load forecasting," *IEEE International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*, 2011.

[45] J. Z. Kolter and J. F. Jr, "A large-scale study on predicting and contextualizing building energy usage," *AAAI Conference on Artificial Intelligence*, 2011.

[46] Z. Aung, M. Toukhy, J. Williams, A. Sanchez, and S. Herrero, "Towards accurate electricity load forecasting in smart grids," *DBKDA*, pp. 5157, 2012

[47] A. Marinescu, C. Harris, I. Dusparic, V. Cahill, and S. Clarke, "A hybrid approach to very small scale electrical demand forecasting," *IEEE PES*, pp. 15, 2014.

[48] M. Frincu, C. Chelmis, M. Noor, and V. Prasanna, "Accurate and Efficient Selection of the Best Consumption Prediction Method in Smart Grids," *IEEE Big Data*, 2014 .

[49] A. Gruenstein, I. McGraw, and I. Badr, "The WAMI Toolkit for Developing, Deploying, and Evaluating Web-Accessible Multimodal Interfaces," *ICMI*, 2008.

[50] Y-S Chang, S-H Hung, N. J. C. Wang, and B-S Lin, "CSR: a Cloud-assisted Speech Recognition Service for Personal Mobile Device," *Conference on Parallel Processing*, 2011.

[51] X. Huang and K. Lee, "On speaker-independent, speaker- dependent, and speaker-adaptive speech recognition," *IEEE Transactions on Speech and Audio Processing*, 1(2):150157, Apr. 1993.

[52] M. Chan, D. Estve, C. Escriba, and E. Campo, "A review of smart homesPresent state and future challenges," *Journal of Computer Methods and Programs in Biomedicine*, 91(1):55-81, Jul. 2008.

[53] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali. "A Review of Smart Homes-Past, Present, and Future," *IEEE Tran. on Systems, Man, and Cybernetics-Part C: Applications, and Reviews*, 42(6), 1190-1203, Nov. 2012.

[54] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29 pp. 16451660, 2013.

[55] A. N. Khan, M. L. M. Kiah, S. U. Khan, and S. A. Madani, "Towards secure mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 23, pp. 12781299, 2013.

[56] B. T. Korel and S. G. M. Koo, "A Survey on Context-Aware Sensing for Body Sensor Networks ," *Journal of Wireless Sensor Network*, vol. 2, pp. 571-583, Aug. 2010.

[57] C. Bettinia, O. Brdiczkab, K. Henricksenc, J. Indulskad, D. Nicklase, A. Ranganathanf, and D. Ribonia, "A survey of context modelling and reasoning techniques," *Journal of Pervasive and Mobile Computing*, 6(2): 161-180, Apr. 2010.

[58] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-Based Activity Recognition," *IEEE Trans. on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, vol. 42, no. 6, Nov. 2012.

[59] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context Aware Computing for The Internet of Things: A Survey," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, May 2013.

[60] M-W Ryu, J. Kim, S-S Lee, and M-H Song, "Survey on Internet of Things: Toward Case Study," *Smart Computing Review*, vo. 2, no. 3, Jun. 2012.

[61] L. Atzori, A. Iera b, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 27872805, 2010.

[62] J. Chong, G. Friedland, A. Janin, N. Morgan, and C. Oei, "Opportunities and Challenges of Parallelizing Speech Recognition," *HotPar*, 2010.

[63] K. You, Y. Lee, and W. Sung, "OpenMP-based parallel implementation of a continuous speech recognizer on a multi-core system," *ICASSP*, 2009.

[64] W-Q Zhang, L. He, Y-L Chow, and R-Z. Yang, "The Study on Distributed Speech Recognition System, " *ICASSP*, 2000.

[65] (2014, Nov. 20). *CMU Sphinx speech library* [Online]. Available: http://cmusphinx.sourceforge.net

[66] (2015, Jan. 13). *Tigen* [Online]. Available: https://www.tizen.org

[67] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, "The HTK Book (for HTK Version 3.4)," *Cambridge University Engineering Department*, Mar. 2009.

[68] (2014, Oct. 31). *Kaldi speech library* [Online]. Available: http://kaldi.sourceforge.net

[69] (2014, Apr. 24). *Open-Source Large Vocabulary CSR Engine Julius* [Online]. Available: http://julius.sourceforge.jp

[70] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in Haystack: Facebooks photo storage, " *OSDI*, 2010.

[71] (2014, Dec. 10). *Apache Hadoop* [Online]. Available: http://hadoop.apache.org

[72] (2014, Dec. 22). *Apache HBase* [Online]. Available: http://hbase.apache.org

[73] M. Anand, E. Ceaser, H. Doddi, C. Goffinet, J. Gudenkauf, and S. Lee, (2012, Nov. 11). *Twitter, Blobstore: Twitters in-house photo storage system* [Online]. Available: `https://blog.twitter.com/2012/blobstore-twitter%E2%80%99s-house-photo-storage-system`

[74] W. Xue. HBase Large Object (LOB) Storage [Online]. Available: `http://files.meetup.com/1350427/HBase_LOB.pdf`

[75] Y-C. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating Skew in MapReduce Applications," *SIGMOD*, 2012.

[76] C. Nyberg. Terasort [Online]. Available: http://sortbenchmark.org

[77] S.-J. Doh and R. M. Stern, *Inter-Class MLLR for Speaker Adaptation*, Proc. IEEE Conf. on Acoustics, Speech, and Sig. Proc., June, 2000, Istanbul, Turkey.

[78] E. B. Gouvea, P. J. Moreno, B. Raj, T. M. Sullivan, and R. M. Stern, "Adaptation and Compensation: Approaches To Microphone And Speaker Independence In Automatic Speech Recognition," *Proceedings of the ARPA Workshop on Speech Recognition Technology*, Harriman, NY, Morgan Kaufmann, D. Pallett, Ed, 1996.

[79] Jetty [Online]. Available: http://www.eclipse.org/jetty

[80] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *EuroSys*, 2010.

[81] *Rabbit MQ Messaging that just works* [Online]. Available: https://www.rabbitmq.com

[82] *Apache Kafka* [Online]. Available: http://kafka.apache.org

[83] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T-M-T Do, O. Dousse, J. Eberle, and M. Miettinen, "The Mobile Data Challenge: Big Data for Mobile Computing Research," *Pervasive Computing*, 2012.

[84] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a Service and Big Data," *Proceedings of the International Conference on Advances in Cloud Computing*, 2012.

[85] Stacey Higginbotham. (2010, September). *Sensor Networks Top Social Networks for Big Data* [Online]. Available: `http://gigaom.com/cloud/sensor-networks-top-social-networks-for-bi-data-2`

[86] BCC Research. (2011, March). *Sensors: Technologies and Global Markets* [Online]. Available: http://www.bccresearch.com/market-research/instrumentation-and-sensors/sensors-technologies-markets-ias006d.html

[87] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Journal of Wireless Communications and Mobile Computing*, 14(18):15871611, Dec. 2013.

[88] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten, "Access Schemes for Mobile Cloud Computing," *MDM*, pp. 387-392, , 2010.

[89] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges," *IEEE Communications Surveys & Tutorials*, 16(1):369-392, May 2013.

[90] S. Abolfazli, Z. Sanaei, E. Ahmed, and A. Gani, "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *IEEE Communications Surveys & Tutorials*, 16(1):337-368, Jul. 2013.

[91] P. Garg and V. Sharma, "Secure Data Storage in Mobile Cloud Computing," *Journal of Scientific & Engineering Research*, 4(4), Apr. 2013.

[92] X. Ma, Y. Zhao, L. Zha, and H. Wang, "When mobile terminals meet the cloud: computation offloading as the bridge," *IEEE Network*, 27(5):28-33, Oct. 2013.

[93] M. Alizadeh and W. H. Hassan, "Challenges and opportunities of Mobile Cloud Computing," *IWCMC*, pp. 660-666, 2013.

[94] G. L. Nemhauser and L. A. Wolsey, "Integer and Combinatorial Optimization," New York, Chichester, Brisbane, Toronto, Singapore, John Wiley and Sons, 1988.

[95] R. Impagliazzo, S. Lovett, R. Paturi, and Stefan Schneider, "0-1 Integer Linear Programming with a Linear Number of Constraints," *CoRR*, Jan. 2014.

[96] R. Impagliazzo and R. Paturi, "The complexity of k-sat," *Journal of Computer and Systems Sciences*, 62(2):367375, Mar. 2001.

[97] R. Schuler, "An algorithm for the satisfiability problem of formulas in conjunctive normal form," *Journal of Algorithms*, 54(1):4044, 2005.

[98] D. Chen, R.G. Batson, and Y. Dang, "Applied Integer Programming: Modeling and Solution," *John Wiley and Sons*, 2010.

[99] M. Junger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. L. Pulleyblank, G. Reinelt, G. Rinaldi, L. A. Wolsey (Eds.), "50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art," *Springer*, 2009.

[100] G. L. Nemhauser and L. A, Wolsey, "Integer and Combinatorial Optimization," *John Wiley and Sons*, 1988.

[101] H. P. Williams, "Logic and Integer Programming," *International Series in Operations Research and Management Science. Springer*, 2009.

[102] L. Wolsey, "Integer Programming," *Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc.*, 1998.

[103] A. Ben-Israel and A. Charnes, "On Some Problems of Diophantine Programming," *Cahiers du Centre dEtudes de Recherche Operationelle*, vol. 4, pp. 215-280, 1962.

[104] F. Glover, "A New Foundation for a Simplified Primal Integer Programming Algorithm," *Operations Research*, vol. 23, pp. 434-451, 1968.

[105] R. D. Young, "A Primal (All Integer), Integer Programming Algorithm," *Journal of Research of the National Bureau of Standards*, vol. 69B, pp. 213-250, 1965.

[106] R. D. Young, "A Simplified Primal (All Integer)," *Integer Programming Algorithm. Operations Research*, vol. 16, pp. 750-782, 1968.

[107] M. W. Padberg, "Ed. Combinatorial Optimization," *Mathematical Programming Study*, vol. 12, 1980.

[108] R. E. Gomory, "An All-Integer Programming Algorithm, in Industrial Scheduling," *J. F. Muth and G. I. Thompson (Eds.), Prentice-Hall*, pp. 193-206, 1963.

[109] F. Glover, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, vol. 13, pp. 879-919, 1965.

[110] F. Glover, "A Pseudo Primal-Dual Integer Programming Algorithm," *Journal of Research of the National Bureau of Standards*, vol. 71B, pp. 187-195, 1967.

[111] A. H. Land and A. G. Doig, "An Automatic Method for Solving Discrete Programming Problems," *Econometrica*, vol. 28, pp. 97-520, 1960.

[112] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. "An Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 11, pp. 972-989, 1963.

[113] E. Balas, "An Additive Algorithm for Solving Linear Programs with Zero-One V ariables," *Operations Research*, vol. 13, pp. 517-546, 1965.

[114] M. Barahona, M. Grotschel, M., G. Junger, "G. Reinelt. An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design," *Operations Research*, vol. 36, pp. 493-513, 1988.

[115] S. Chopra, E. Gorres, and M. R. Rao, "Solving the Steiner Tree Problem on a Graph Using Branch and Cut," *ORSA Journal on Computing*, vol. 4, pp. 320-335, 1992.

[116] M. Grotschel, C. L. Monma, and M. Stoer. "Computational Results with a Cutting Plane Algorithm for Designing Communication Networks with Low-Connectivity Constraint," *Report No 187, Schwerpunktprogramm der Deutschen Forschungsgemeinschaft, Universitat Augsburg*, 1989.

[117] T. L. Magnanti and R. Vachani, "A Strong Cutting Plane Algorithm for Production Scheduling with Changeover Costs," *Operations Research*, vol. 38, pp. 456-473, 1990.

[118] Y. Pochet and L. A. Wolsey, "Solving Multi-Item Lot Sizing Problems Using Strong Cutting Planes," *Management Science*, vol. 37, pp. 53-67, 1991.

[119] G. Desaulniers, J. Desrosiers, and M. Solomon (Eds.). "Column Generation," *Kluwer*, 2005.

[120] R. J. Dakin, "A Tree Search Algorithm for Mixed Integer Programming Problems," *Computer Journal*, vol. 8, pp. 250-255, 1965.

[121] E. M. L. Beale and J. A. Tomlin. "Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables," *In: Proc. of the Fifth International Conference on Operational Research, J. Lawrence, (Ed.), Tavistock Publications*, pp. 447-454, 1970.

[122] E. M. L. Beale and J. J. H. "Forrest. Global Optimization Using Sprcial Ordered Sets," *Mathematical Programming*, vol. 10, pp. 52-69, 1976.

[123] A. W. M. Dress and W. Wenzel, "A New Look at the Greedy Algorithm," *Appl. Math. Lett.*, vol. 3, pp. 33-35, 1990.

[124] K. Murota, "Combinatorial Relaxation Algorithm for the Maximum Degree of Subdeterminants: Computing Smith-McMillan Form at Infinity and Structural Indices in Kronecker Form," *Appl. Algebra Engin. Comm. Comput.*, vol. 6, pp. 251-273, 1995.

[125] E. A. Silver, "An Overview of Heuristic Solution Methods," *Journal of the Operational Research Society*, col. 55, pp. 936-956, 2004.

[126] K. Dowsland, "Simulated Annealing," *Modern Heuristic Techniques for Combinatorial Problems, C. R. Reeves (Ed.). Blackwell*, pp. 20-69, 1993.

[127] S. Kirkpatrick, C. Gellat, and M. Vecchi. "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[128] M. Pirlot, "Heuristic Search Methods," *Oper. Res. Designing Practical Solutions; Tutorial and Research Review Papers, Euro XIII/OR 36, The Joint EURO/Oper. Res. Society Conference; University of Strathclyde, Glasgow*, pp. 180-201, July, 1994.

[129] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research*, 13(5):533-549, 1986.

[130] H. Osman and G. Laporte, "Meta-Heuristics: A Bibliography," *Annals of Operations Research*, vol. 63, pp. 513-628, 1996.

[131] E. Aarts and J. K. Lenstra, "Local Search in Combinatorial Optimization," *John Wiley and Sons*, 1997.

[132] C. Voudouris and E. Tsang, "Guided Local Search," *International Series in Operations Research and Management Science, Kluwer Academic Publishers*, 2002.

[133] H. R. Lourenco, O. Martin, and T. Sttzle, "Iterated Local Search," *International Series in Operations Research and Management Science*, vol. 57, pp. 321-353, 2002.

[134] P. Hansen, N. Mladenovi, and J. A. Moreno Perez, "Variable Neighborhood Search: Methods and Applications," *Annals of Operations Research*, vol. 175, pp. 367-407, 2010.

[135] E. M. L. Beale and J. A. Tomlin, "Special Facilities in a General Mathematical Programming System for Nonconvex Problems Using Ordered Sets of Variables," *Proc. of the Fifth International Conference on Operational Research, J. Lawrence, (Ed.), Tavistock Publications*, pp. 447-454, 1970.

[136] F. Glover, "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, vol. 8, pp. 156-166, 1977.

[137] A. Colorni, M. Dorigo, and V. Maniezzo. "Distributed Optimization by Ant Colonies," *Proc. of the 1st European Conference on Artificial Life, F. J. Varela and P. Bourgine (Eds.), Cambridge, MA, MIT Press*, pp. 134-142, 1991.

[138] R. Eberhart and Y. Shi. "Particle Swarm Optimization: Developments, Applications and Resources," *Proc. Congr. Evolutionary Computation*, vol. 1, pp. 81-86, 2001.

[139] P. Moscato, "Memetic Algorithms," *Handbook of Applied Optimization, P. Pardalos and M. Resende (Eds.), Oxford University Press*, pp. 157-167, 2002.