Communication Infrastructures for Cloud Computing

Hussein T. Mouftah
University of Ottawa, Canada

Burak Kantarci
University of Ottawa, Canada

A volume in the Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director: Lindsay Johnston
Production Manager: Jennifer Yoder
Publishing Systems Analyst: Adrienne Freeland
Development Editor: Christine Smith
Acquisitions Editor: Kayla Wolfe
Typesetter: John Crodian
Cover Design: Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Library of Congress Cataloging-in-Publication Data

Communication infrastructures for cloud computing / Hussein T. Mouftah and Burak Kantarci, editors.
pages cm
Includes bibliographical references and index.
ISBN 978-1-4666-4522-6 (hardcover) -- ISBN (invalid) 978-1-4666-4523-3 (ebook) -- ISBN 978-1-4666-4524-0 (print & perpetual access) 1. Cloud computing. 2. Internetworking (Telecommunication) I. Mouftah, Hussein T., editor. II. Kantarci, Burak, 1981- editor.
QA76.585C66 2013
004.67'82--dc23

2013023985

This book is published in the IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC) (ISSN: 2327-3453; eISSN: 2327-3461)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 8
Accelerating Mobile-Cloud Computing: A Survey

Tolga Soyata
University of Rochester, USA

Wendi Heinzelman
University of Rochester, USA

He Ba
University of Rochester, USA

Minseok Kwon
Rochester Institute of Technology, USA

Jiye Shi
UCB Pharma, UK

ABSTRACT

With the recent advances in cloud computing and the capabilities of mobile devices, the state-of-the-art of mobile computing is at an inflection point, where compute-intensive applications can now run on today's mobile devices with limited computational capabilities. This is achieved by using the communications capabilities of mobile devices to establish high-speed connections to vast computational resources located in the cloud. While the execution scheme based on this mobile-cloud collaboration opens the door to many applications that can tolerate response times on the order of seconds and minutes, it proves to be an inadequate platform for running applications demanding real-time response within a fraction of a second. In this chapter, the authors describe the state-of-the-art in mobile-cloud computing as well as the challenges faced by traditional approaches in terms of their latency and energy efficiency. They also introduce the use of cloudlets as an approach for extending the utility of mobile-cloud computing by providing compute and storage resources accessible at the edge of the network, both for end processing of applications as well as for managing the distribution of applications to other distributed compute resources.

DOI: 10.4018/978-1-4666-4522-6.ch008

INTRODUCTION

Recent developments in mobile computing have truly empowered human users, as mobile computing can augment cognitive capabilities dramatically, e.g., through voice recognition, natural language processing, machine learning, augmented reality, and decision-making (Satyanarayanan et al., 2009). With recent advances in mobile devices, coupled with the technological advances in wireless and cloud technologies, computationally intensive applications can now run on devices with limited resources such as tablets, netbooks and smartphones using the cloud remotely as an additional computational resource.

Although different definitions exist in the literature (Dinh, et al, 2011; Fernando et al., 2013), we define mobile-cloud computing as the co-execution of a mobile application within the expanded mobile/cloud computational platforms to optimize an objective function. A typical objective function is the *application response time*, where the goal is to minimize the objective function. Expanding the application computational resources beyond the mobile is necessary for applications where the objective function cannot be minimized sufficiently by the mobile platform alone (e.g., real-time face recognition), as well as for applications that rely on data not stored on the mobile device. In mobile-cloud computing, it is crucial to provide the user seamless, transparent and cost-effective services as mobile devices rent computing, storage, and network resources from the cloud in order to process and store a vast amount of data (AWS, 2012; Microsoft, 2012; Google, 2012). (AWS, 2012)(Microsoft, 2012) (Google, 2012)

We define *application cost* as an example objective function that quantifies the fees charged by Cloud operators, such as Amazon Web Services, during the execution of the application. For example, Amazon charges for compute-usage per hour per CPU instance, which implies increasing application costs as the required amount of computation increases. Similarly, cloud operators charge for the usage of database instances, such as Microsoft SQL Server. Table 1 shows some example mobile-cloud applications and their computational/storage demands, as well as their application response-time sensitivity. While applications requiring higher computational and storage resources might cost more during operation in a Cloud platform such as AWS, certain response-time sensitive applications, such as the Battlefield application described in Table 1, might tolerate this increased cost due to their need for low response time. Notice that Cloud operators charge less for compute-resources with lower response time guarantees. Specifically, while AWS charges nothing for *Micro* instances with *no* response time guarantees, it charges a small amount for the *Small* instance, and significantly higher for the *Large* instance, which is a dedicated CPU instance. By the preparation of this document, the AWS pricing for these instances ranged from $0.10 to $0.40 per core per GHz per hour (AWS, 2012), where the unit price decreased with a higher core-count commitment (i.e., number of cores available to an instance). This implies a rich variety of options when executing mobile-cloud applications. The choice of the Cloud CPU instances depends on the application priorities listed in Table 1.

The primary focus of this chapter is to elaborate on the techniques that enable these mobile-cloud applications to achieve the goals listed in Table 1. Although the demands of these applications will not change from that shown in this table, achieving certain goals might never become possible by using mobile-only or even a mobile-cloud combination. This is due to the limited computation and storage on a mobile device, which does not permit the processing or storage of large amounts of data locally, as well as the high network latencies connecting the mobile and cloud, plac-

Table 1. Cloud-based applications and their resource requirements. Each application has a significantly different response time requirement and resource utilization tolerance to reduce costs while still keeping the functionality within expected bounds.

Application	Description	Database Size	Compute Resources	Response Time Sensitivity
Battlefield	Assist soldiers in the battlefield through real-time object recognition	HIGH	HIGH	HIGH
Natural Language Processing	Perform real-time speaker or speech recognition	LOW	MEDIUM	MEDIUM
Airport	Conduct real-time face recognition of known criminals	HIGH	HIGH	HIGH
Fire	Assist fire fighters with disaster in real-time	MEDIUM	MEDIUM	MEDIUM
Medicine	Accelerate medical research (e.g., recognizing DNA sequences in real-time from a microscope while the research is in progress)	HIGH	MEDIUM	MEDIUM
Archeology	Recognize archeological structures in real-time while the researchers are at the search site	HIGH	LOW	LOW
Surgery	Recognize issues (e.g., tumors) in real-time from a cloud-based medical database while the surgery is in progress	HIGH	MEDIUM	HIGH
Amber Alert	Identify criminals by searching through the FBI database to match a photo taken by a camera	LOW	LOW	MEDIUM
Social Network	Profile online users by searching through a database for marketing purposes	HIGH	LOW	LOW

ing a lower bound on application response times when utilizing the cloud for processing and storage of large amounts of data. Later in this chapter, we will describe how the required application response times may be achieved by using an edge-server device called a "cloudlet," creating a mobile-cloudlet-cloud platform.

This chapter is organized as follows: First, the technological challenges and the state-of-the-art in computational and storage capabilities of mobile devices and the network latencies are studied. Issues related to energy efficiency and security are also explored, followed by a brief study of the aforementioned intermediate layer cloudlet and its function in the mobile-cloud computing environment. Existing architectural designs as well as performance enhancement techniques proposed in the literature for mobile-cloud as well as mobile-cloudlet-cloud computing are surveyed. The chapter is concluded with discussions on future research areas.

TECHNOLOGICAL CHALLENGES IN MOBILE-CLOUD COMPUTING

Running the resource-intensive applications enumerated in Table 1 far exceeds the capabilities of today's mobile devices. The constraints on mobile devices in terms of weight, size, battery life, ergonomics, and heat dissipation limit the resources available in mobile hardware, including the processor speed, memory size, and storage capacity. Given these challenges, mobile computing benefits tremendously when combined with cloud computing that can offer virtually limitless computing power and storage space, as well as access to up-to-date databases, only available in the cloud.

There are, however, several technical obstacles to enabling mobile devices to benefit from cloud computing resources, including the compute capability and storage capacity available at the mobile, network connectivity and latency challenges, the need for energy-efficiency at the mobile device,

and security concerns. As each one of these constraints affect mobile-cloud computing in a unique way, they will be individually detailed in the following subsections.

Compute Capability and Storage Capacity

Despite an order of magnitude higher computational power of today's mobile devices compared to the ones from just a few years ago, the relative computational power ratio of a non-mobile and a mobile device is likely to stay approximately the same in the foreseeable future. This is due to the architectural and technological state-of-the-art advances being applied to mobile platforms as well as non-mobile platforms simultaneously by different market leaders such as Intel for desktop platforms and ARM for mobile platforms. The most important metric, computer-power-per-Watt (also defined as GFLOPS-per-Watt) has almost reached equal levels in both mobile and desktop platforms. For example, a Tegra3-based mobile phone incorporating an ARM CPU and an Nvidia GPU at the core can deliver approximately 10 GFLOPS/Watt (Tegra3). Alternatively, a desktop platform composed of an INTEL Core i7 CPU and an Nvidia Geforce 600 GPU (GeForce600, 2012) nearly has the same power efficiency metric, delivering around 10 GFLOPS/Watt compute power. This is due to the significant recent advancements in mobile processors: almost every power efficiency technique employed in desktop CPUs is now being incorporated into mobile CPUs, with the most important being the ability to architect the CPU with multiple cores, which is known to have a dramatic energy reduction advantage (Guo et al., 2010).

The storage technology is slightly different in that the widespread use of Solid State Disks (SSDs) allowed mobile devices to be built with storage capacities that are currently around 64GB to 128GB. This is currently an order of magnitude less than that for desktop platforms, which enjoy inexpensive hard disks in the multiple-TB range. This means that mobile-cloud applications that require significant local data storage in the mobile are not feasible.

Network Connectivity

A primary concern in the use of mobile-cloud computing is the non-negligible latency over the WAN (Wide-Area Network) between the mobile and the cloud, which hurts the user experience in mobile-cloud computing. Interactive applications that constantly engage the users are likely to suffer the most from long delay, jitter and jerky and sluggish processing. Studies (Satyanarayanan et al., 2009) show that the quality of client performance becomes highly variable with long latency.

In order to measure latencies over WAN connections, we ran a simple program that sends ping packets from a client computer to servers in cloud datacenters in January and February 2012. The client computer was located in Rochester, New York, in the United States, and we used the five datacenters available in AWS (AWS, 2012), which are all located in geographically different regions, namely in Virginia and Oregon in the United States, Ireland in Europe, São Paulo in South America, and Singapore in Asia.

Table 2 shows the mean and standard deviation of these latencies for the AWS datacenters when being accessed by the client computer from wired and wireless networks. This data clearly indicates the challenges in running a mobile-cloud application that uses the AWS datacenters as cloud servers. The response time of such an application will be lower-bounded by the mean latency of the communication to the datacenter it is using as cloud servers. Alternatively, the predictability of the response time will be determined by the standard deviation of the latency. While there have been significant improvements in network throughputs over the past decade, allowing users to enjoy such high-speed connections with 50 Mbps downstream bandwidth (e.g., DOCSIS3

Table 2. Average and standard deviation of latencies over wireless connections (ms)

Wireless Connections	**Wi-Fi**					**3G**				
Datacenter	VA	OR	Ireland	São Paulo	Singapore	VA	OR	Ireland	São Paulo	Singapore
Mean	253	389	293	434	697	930	817	798	872	1061
Std Deviation	470	635	520	704	1278	595	710	915	1079	2060
Wired Connections	Weekend					Weekday				
Datacenter	VA	OR	Ireland	São Paulo	Singapore	VA	OR	Ireland	São Paulo	Singapore
Mean	122	322	294	389	580	42	223	196	389	546
Std Deviation	124	525	201	166	242	18	41	25	166	34

cable standard [DOCSIS, 2012]), network latencies have not improved nearly at the same rate.

Although the network latencies might improve in the future, this is expected to be at a much slower pace, potentially keeping the latencies observed in Table 2 approximately the same in the foreseeable future. For example, an application requiring a 200 ms response time (close to what can be described as *real-time*) is not a candidate to run on cloud servers residing across international boundaries with 300 ms to 1100 ms latencies. This presents a dilemma in mobile applications in that the particular emphasis should be placed on latency, not the throughput, when developing an application. Alternatively, any intermediate device, such as the cloudlet that will be described later, should be targeted to reduce the negative impact of this high latency.

Power and Energy Consumption

Today's mobile devices incorporate significantly sophisticated power management circuitry (Qualcomm, 2012). This, combined with power consumption demands that change drastically in sudden peaks imply a sophisticated power consumption pattern from the mobile device based on the activities being performed (i.e., talk, compute, or run applications). Power consumption could change between sudden peaks of mW and a few W. While the power consumption is in fact an irrelevant measure in terms of battery life, the energy consumption is the relevant measure in determining the battery life.

To quantify the utility of mobile-cloud computing, one must take into account the energy demands of computation and communication separately. Analyzing these two activities separately will shed light onto the balance that must be maintained between computation and communication via efficient scheduling. In the following two subsections, we study the power and energy demands of these two activities.

1. **Computation Power and Energy Consumption:** Since the *computation energy* is the only relevant metric for determining battery life, we analyze the amount of energy required to execute an identical task in a desktop and mobile platform. The *energy efficiency* metric, defined as GFLOPS-per-Watt describes how many Watts of power is consumed to while delivering 1 GFLOPS of computational output. This metric is 10 GLOPFS/Watt in a modern mobile processor such as Tegra 3 (Tegra, 2012), while it is almost in the same range for a modern CPU/GPU-based desktop computer (GeForce600, 2012). Alternatively, a mobile device operates at around one Watt average power consumption, whereas a desktop platform could reach 200-1000 Watts of power consumption.

Although a given computational task (e.g., face recognition) may consume more power and execute in a shorter amount of time on a non-mobile platform (e.g., PC), it will consume less power on a mobile platform (strictly due to the aforementioned constraint on peak power), and is, therefore, expected to complete in a longer time period. However, due to the fact that approximately the same amount of computational energy is required for the same task, the mobile platform will take two to three orders-of-magnitude longer to execute the same task, since its peak power output is nearly two to three orders-of-magnitude lower.

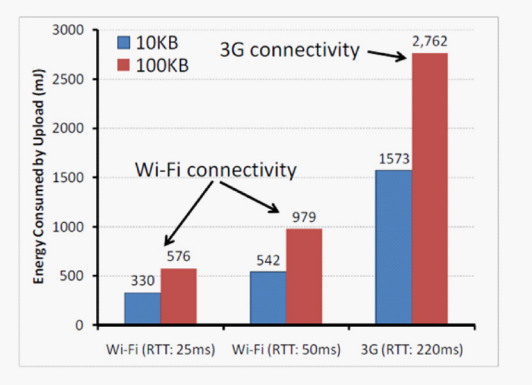
2. **Communication Power and Energy Consumption:** The communications energy of Wi-Fi and 3G are depicted in Figure 1. As shown in this figure, 3G requires much higher energy levels due to its inability to transfer large amounts of data, while Wi-Fi can transfer almost an order-of-magnitude more data within the same power envelope. Based on this figure, 3G connections require 2,762 mJ per 100KB (i.e., 27.62 $\mu J/B$). Alternatively, Wi-Fi requires around 5 to 10 $\mu J/B$, making it more energy efficient. These different energy profiles suggest that, when determining optimum algorithms that partition computation and communication, the energy patterns of both must be considered. As an example, assume that an algorithm has a choice among different computation vs. communication options as presented below:
    a. **Case 1: Front-loading.** In this case, most of the computation is done on the Tegra 3 mobile device that has a 10 GFLOPS compute-capability, and a 10 GFLOPS/W energy efficiency, and 3G is used for communication. For the Case 1 algorithm, 20 GFLOP of computation and 100KB of data transfer are necessary. Total energy consumption is 2,000 mJ (i.e., 20 GFLOPS / (10 GFLOPS/Watt) = 2 Watt * 1 second = 2J = 2,000 mJ) for the computation (based on the aforementioned 10 GFLOPS/W for Tegra 3) and 2,762 mJ for the 3G communication (from Figure 1), yielding a total energy demand of 4,762 mJ for the entire task.
    b. **Case 2: Back-loading.** Assume now that the same algorithm can be modified to perform more of the computation in the cloud at the expense of increased data transfer via 3G. In this Case 2, the computation in the mobile will be halved to 10 GFLOP at the expense of doubled data transfer size to 200 KB. This will create a compute-energy demand of 1,000 mJ and a communication demand of 5,524 mJ, resulting in a total energy demand of 6,524 mJ, clearly an unfavorable choice.
    c. **Case 3: Back-loading on a faster communications link.** If we consider Case 2 on a faster Wi-Fi link (e.g., 576 mJ/100KB as shown in Figure 1, the energy demand for the front-loading and back-loading cases are (2,000+576=2,576 mJ) and (1,000+1,152=2,152 mJ), respectively, making back-loading a better alternative for faster Wi-Fi links, although the decision is the reverse for 3G links.

Security

When considering outsourcing the computational tasks of a mobile application to the cloud, an important issue arises for certain applications: the security of the data being transmitted/received by the application. Depending on the application, the security of the data carries a varying importance. For example, for online games being played by a few gamers through the mobile device, the security factor is negligible, while for remote health assistance applications, it is of utmost importance.

Due to the emergence of applications using wireless sensors with built-in low-power micro-

Figure 1. The energy consumption of Wi-Fi connectivity vs. 3G connectivity (reprinted from Cuervo, et al., 2010, with permission of the authors)



controllers and the sudden spike in interest for concepts such as Internet-of-Things (IOT, 2012), the security of the data being transported by the application has become one of the most important concepts to consider. Additionally, emerging telemedicine applications also emphasize the importance of security and data privacy (HIPAA, 1996) within the mobile-cloud platforms (Pattichis et al., 2002), (Varshney, 2007), (Wood et al., 2008). The concern for data security is of great importance for mobile devices with powerful processors with a 1 W power budget (e.g., Tegra3, 2012), but it is particularly challenging for embedded processors with only a mW power envelope, such as the Microchip 32-bit microcontroller family (PIC32, 2012). This is due to the fact that the encryption of the data using standard Advanced Encryption Standard (AES) encryption (NIST, 2001; AES, 2012) is compute-intensive and strains the computational resources of the underlying computational platform unless specialized crypto-accelerators are used. However, as most of today's devices have AES hardware acceleration built-in, the encryption time and energy is typically less than the transmission energy of the data, thereby making encryption widely available with minimal impact on cost and energy.

Cloud and Cloudlet: Addressing These Challenges

With the help of the cloud, mobile devices may be able to offload the computationally-intensive parts of their applications. The enormous resources of the cloud may minimize the time and energy cost of the mobile applications on those computations. However, as described in the Microsoft MAUI project (Cuervo et al., 2010), some applications might never be feasible from mobile devices, due to the high latency mobile-cloud connection. Adding a *cloudlet*, a local device that provides 100 to 1000 times higher computational power than the mobile device with minimal latencies, creates possibilities for running latency sensitive and computationally-intensive applications from a mobile device (Satyanarayanan et al., 2009). The notion of a cloudlet was introduced as a means to overcome some of the technical obstacles described above. The main idea is to provide the abundant resources needed at mobile devices not from distance clouds, but from a nearby cloudlet.

As Satyanarayanan et al. (2009) point out, the key differences between the cloudlet and a conventional cloud are listed in Table 3. Note that soft state refers to cache copies of data or code that are available elsewhere (e.g., mobile device or the cloud), whereas hard state refers to sole copy of data or code. Since a cloudlet only contains soft state, the loss of a cloudlet is not catastrophic. Cloudlets allow offloading a portion of the tasks or changing the timing of the execution of the subtasks to speed up the execution of the overall task by using synchronous optimization techniques (Soyata et al., 1993; Soyata & Friedman, 1994; Soyata et al., 1995; Soyata, 1999) as well as a re-shaping of the network traffic by aggregating network packets. With proper task management algorithms, a cloudlet may be able to leverage the power of distant cloud servers to maximize the performance while minimizing the impact of long network latency.

Table 3. The major differences between the cloudlet and a conventional cloud (reprinted from Satyanarayanan, et al., 2009, with permission of the authors)

	Cloudlet	Cloud
State	Only soft state	Hard and soft state
Management	Self-managed, little or no professional attention	Professionally administered 24/7
Environment	"Datacenter in a box" at business premises	Room with power conditioning and cooling
Ownership	Decentralized ownership by local business	Centralized ownership by Amazon, Yahoo etc ...
Network	LAN latency/bandwidth	Internet latency/bandwidth
Sharing	Few users at a time	Hundreds to thousands of users at a time

In the following section, different approaches will be presented that use the cloudlet as a buffering layer to either speed up the computation or to reduce the negative effect of the communication latency to the cloud.

ARCHITECTURAL DESIGN

As stated in the previous section, in mobile cloud computing, there is a clear need for a mechanism to handle the interoperations between the mobile device and the cloud servers in order to improve the performance. Depending on whether or not cloudlets are used, the current research on the design of mobile-cloud architectures can be categorized as cloudlet architectures and non-cloudlet architectures. Many of the technologies being used in non-cloudlet architectures can also be adapted to cloudlet architectures. In this section, we introduce the state-of-the-art for these mobile-cloud and mobile-cloudlet-cloud architectures.

Platforms Providing Cloud Services

The "cloud" may consist of commercial servers like the Amazon Web Services (AWS, 2012), Microsoft's Windows Azure (Microsoft, 2012), and the Google Cloud Platform (Google, 2012), or it may be created ad hoc from available computing resources, as shown through recent studies (Ali, 2009). While ad hoc clouds have been conventionally created from high-end servers or desktop platforms, recently mobile platforms have been explored as a source for the computing resources.

For example, Hyrax (Marinelli, 2009) demonstrated the concept of using smartphones as a cloud of computing resources. Marinelli developed a mobile-cloud computing system named Hyrax by porting Hadoop Apache, an open-source implementation of MapReduce, to Android smartphones. Hyrax allows computing jobs to be executed on networked Android smartphones. However, the performance of Hyrax was poor compared with Hadoop on traditional servers, not only because the smartphones were much slower at that time, but also because Hadoop was not originally designed, nor optimized, for mobile devices.

GEMCloud (Ba et al., 2013) is another example of using mobile devices to create an ad hoc cloud of computing resources. By utilizing distributed mobile devices to cooperatively accomplish large parallelizable computational tasks, Ba *et al.* envision that such approaches can make use of the massive amount of idle computing power that is potentially available to the public. More importantly, the authors show that a mobile computing system like GEMCloud has significant advantages in energy efficiency over traditional desktop cloud servers when the overall system is considered, rather than each individual computational device (e.g., mobile).

Other examples of ad hoc cloud systems are NativeBOINC (BOINC, 2012) and BOINC Mobile (Eastlack, 2011), Android platform (Android, 2012) equivalents of the BOINC volunteer computing platform originally designed for PCs and game console platforms (Anderson, 2004). Since the physical devices that build up the cloud determine the cloud's characteristics such as computing power, energy efficiency and network latency, it is important to profile the cloud servers and take this into account when designing the mobile-cloud computing system.

Mobile-Cloud Architectures

Mobile-cloud computing has been investigated since shortly after the concept of cloud computing was introduced in mid-2007, and it has attracted great interest in the research community (Dinh et al., 2011). Some important implementations of mobile-cloud computing, including MAUI (Cuervo et al., 2010), CloneCloud (Chun & Maniatis, 2009; Chun et al., 2011) and Virtual Smartphone over IP (Chen et al., 2012; Chen & Itoh, 2010) employ cloud servers to process application partitions offloaded by the mobile device. As discussed previously, the cloud servers may be located in a commercial cloud or an ad hoc cloud.

Although architectural details may vary in different mobile-cloud computing implementations, some common components are often included on top of the operating system and hardware layers:

- A *Partitioner* that analyzes the application and determines which part(s) of the application can be offloaded to the cloud. Depending on the technique being used, the partitioning granularity may be application-level, thread-level, method-level or even line-of-code-level. For applications that cannot be partitioned, the *Partitioner* is not necessary.
- A *Profiler* that collects the mobile device's system measurements to identify the performance status, resource status and other contextual information. The performance measurements may include network condition (e.g., type of network being used, signal strength, bandwidth, computing power and response latency of various cloud services, etc.), screen brightness, CPU, memory and storage usage. The resource status may include remaining battery, available computing power (derived from CPU, memory usage), and the resources required by the application (or method, thread, depending on the granularity offered by the *Partitioner*). Other contextual status may include location, acceleration, temperature, date and time, etc.
- A *Solver* that gathers information from the *Partitioner* and the *Profiler* to decide how to offload the partitions to the cloud based on an optimization algorithm.
- A *User Agent* on the mobile device and a *Coordinator* on the server that handle the authentication and security. The *Coordinator* may also interact with the server database that stores the mobile device users' profiles (e.g., device specifications, user configurations, subscribed services, contextual data) and activity logs. The *Coordinator* allocates the resources on the cloud server for the mobile users.

In general, MAUI (Cuervo et al., 2010), CloneCloud (Chun & Maniatis, 2009; Chun et al., 2011) and the Virtual Smartphone over IP (Chen & Itoh, 2010; Chen et al., 2012) architectures all have the above components or components with similar functionalities.

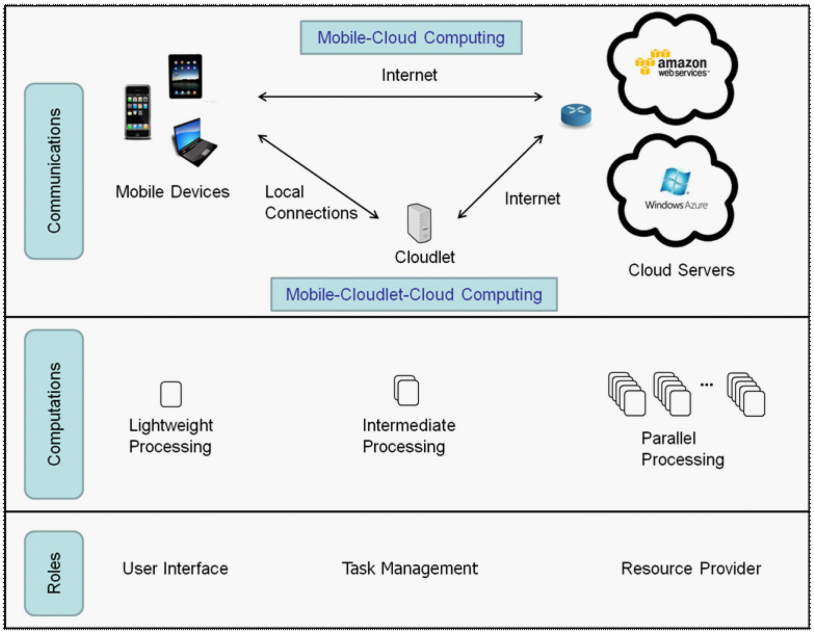
Cloudlet Architectures

As described in Section II, cloudlets can be used as intermediaries between mobile devices and cloud servers. One of the first implementations of a cloudlet architecture was a prototype mobile-

cloudlet computing system named Kimberley (Satyanarayanan et al., 2009), developed by Satyanarayanan *et al.* The authors envisioned a cloudlet as a "data center in a box" widely dispersed throughout the Internet. Unlike the cloud, the cloudlet is self-managed with decentralized ownership, maintains only soft states, is connected to the mobile over a LAN, not a WAN, and is accessed by only a few users at a time. As a proof-of-concept implementation, the Kimberley system utilizes a local server as a cloudlet to process application partitions. The authors show that in some cases, a local server is able to provide enough computing power to boost the execution speed of a mobile application, while in other cases, the computing resources provided by a local server may not be enough and a cloud has to be used to fulfill the computation and storage requirements of the mobile application. Figure 2 elaborates the cloudlet architecture in comparison with the direct mobile-cloud architecture. The major role of a cloudlet in a mobile-cloudlet-cloud network is task management. It may also help the mobile with some intermediate processing.

The MOCHA (Mobile Cloud Hybrid Architecture) architecture was created as a solution to massively-parallelizable mobile-cloud applications (Soyata et al., 2012a, 2012b) by Soyata *et al*. In MOCHA, mobile devices such as smartphones, touchpads, and laptops are connected to the cloud via a cloudlet, a dedicated device designed from commodity hardware supporting multiple network connections such as 3G/4G, Bluetooth, and WiFi. The cloudlet determines how to partition the computation among itself and multiple servers in the cloud to optimize the overall quality of service (QoS) based on continuously updated statistics of the QoS metrics (e.g., latency, cost) over the different links/routes. The authors demonstrate the concept of MOCHA via a mobile-cloud application demanding real-time response, such as face recognition (Soyata

Figure 2. The mobile-cloud computing and mobile-cloudlet-cloud computing architectures: mobile devices directly interact with a cloud or via the cloudlet and use dynamic partitioning to achieve their quality of service (QoS) goals (e.g., latency, cost)



et al., 2012a) and simulate the same architecture in a battlefield application where the response time is of primary importance (Soyata et al., 2012b).

Similar to the characteristics of cloudlets assumed by the Kimberley (Satyanarayanan et al., 2009) and MOCHA (Soyata et al., 2012b) architectures, a two-level architecture is introduced by Ha *et al.* (Ha et al., 2012). This two-level architecture leverages both today's unmodified cloud infrastructure (Level 1) and a second level data center, named 1WiFi, at the edge of the Internet (Level 2), servicing currently-associated mobile devices. The Level 2 data centers are powerful, well-connected and safe cloudlets that only have cached soft state from Level 1 data centers or buffered data from mobile devices. Trust issues and speed of provisioning are the new challenges to this architecture and must be investigated before it can be widely deployed.

In some scenarios such as an office building, multiple cloudlets may be located closely to each other and may be connected in a peer-to-peer fashion. In this case, routing among the cloud servers, the cloudlets and the mobile devices has to be considered. In (Fesehaye et al., 2012), Fesehaye *et al.* propose two types of routing schemes, namely distributed and centralized routing. In distributed routing, the routing table is constructed and maintained by the cloudlets. The cloudlets periodically broadcast their presence information to the neighboring nodes and the other cloudlets. When a mobile user hears a broadcast message from a cloudlet, it records the latest cloudlet ID into its cloudlet table. Each mobile user also periodically broadcasts its ID to let the cloudlet in range register this user and forward it to other cloudlets. In centralized routing, the central server is responsible for constructing and maintaining the routing table. The cloudlet periodically sends the IDs of its mobile users, its own ID and its neighboring cloudlets' IDs to the central server. The central server then computes the routing table for each cloudlet and installs the forwarding tables into the cloudlets.

Simulations were conducted to evaluate the performance of this architecture. In the simulations, distributed routing was chosen as the routing scheme for the cloudlet architecture. The results show that the cloudlet-based approach has lower data transfer delay and higher content delivery throughput than the cloud-based approach. The results were under the assumption that the WiFi transmission range is larger than $250m$. Therefore, the author suggests using the latest technologies such as Flashlinq (Corson et al., 2010) or by using Wi-Fi repeaters to achieve a desired coverage. Since there are no performance comparisons with the centralized routing, questions still remain as to which routing scheme has better performance.

TASK MANAGEMENT AMONG MOBILE, CLOUDLET, AND CLOUD

The goal of developing a cloudlet-assisted mobile-cloud computing system is to improve the performance (e.g., latency, energy efficiency, monetary cost) on the mobile device. One important approach for improving the performance is to offload partial or full execution of the application to the more resourceful cloudlet or the cloud (Karthik Kumar et al., 2013). Shifting the computation load to a communication load may lead to substantial gains in performance.

The computation offloading approaches are based on virtual machine technologies and can be viewed as middleware designs. Besides the middleware that enables the code offloading, task distribution algorithms and control policies are needed to improve the performance to its best. In this section, we will introduce both the middleware designs and the task distribution algorithms that enhance the performance of mobile computing.

Computation Offloading Approaches

Despite the advances in mobile device technologies, the processing and storage capabilities of mobile devices are still not comparable to those of

servers (or the cloudlet) and will continue to lag in the near future. In order to run computationally-intensive applications, the mobile can offload some of the computation to servers while the mobile device computes only lightweight parts of the application. A Virtual Machine (VM) can support individual processes or a complete system running on flexible hardware platforms, thereby providing the feasibility to migrate partial or full applications from the mobile device to more powerful cloudlet/cloud servers without major modifications to the application. Therefore, the application processing time can be shortened while the energy consumption on the mobile device is reduced. Yet this approach poses several technical challenges. First, how can we identify and partition the compute-intensive or energy-hungry parts within the mobile application automatically? Second, what strategy should a mobile device employ for partitioning and offloading with the goal of minimizing computation time and maximizing energy savings? Third, how can we implement such a system from a practical point of view?

In this subsection, we provide an overview of the state-of-the-art VM-based techniques for mobile-cloudlet/cloud computing. These include 1) an approach employed by the Kimberley system (Satyanarayanan et al., 2009) that demonstrates the feasibility of VM synthesis using VirtualBox, 2) an approach used in MAUI (Cuervo et al., 2010) that provides both full and fine-grained remote execution using the .Net framework, 3) the technique in CloneCloud that supports thread granularity partitioning (Chun & Maniatis, 2009; Chun et al., 2011) using Dalvik VM, 4) an approach by Chen *et al.* (Chen & Itoh, 2010; Chen et al., 2012) that enables offloading on non-customized Android devices also using Dalvik VM, and 5) an OSGi approach by Verbelen *et al.* (2012). Table 4 compares these five approaches discussed in this subsection.

1. **Virtual Box in Kimberley:** Satyanarayanan *et al.* implemented a VM for the Kimberley architecture (Satyanarayanan et al., 2009) prototype using a technique called dynamic Virtual Machine synthesis that employs transient cloudlet customization. A small VM overlay is delivered by a mobile device to the cloudlet infrastructure, which creates and launches the VM using a base VM plus the delivered VM for the application. The prototype was implemented on a Nokia N810 tablet running Maemo 4.0 Linux, and the cloudlet infrastructure was implemented using Ubuntu Linux. Kimberley uses VirtualBox as the VM manager and a tool called "Kimberlize" to create VM overlays and synthesize those overlays with base VMs to create a launchVM. Both the mobile and

Table 4. Comparison of task partitioning approaches from Kimberley (Satyanarayanan, et al., 2009), MAUI (Cuervo, et al., 2010), CloneCloud (Chun & Maniatis, 2009; Chun, et al., 2011), Chen's approach (Chen & Itoh, 2010; Chen, et al., 2012), and Verbelen's approach (Verbelen, et al., 2012)

Publication	Technologies	Platform	Granularity	Application Development Difficulty
Kimberley	VirtualBox	Linux	Application	Low
MAUI	.NET framework	Windows	Method	Requires application developer's annotations
CloneCloud	Dalvik VM	JavaVM supported	Thread	No annotation required
Chen's approach	Dalvik VM	JavaVM supported	Thread	No annotation required
Verbelen's approach	OSGi	JavaVM supported	Component	Requires application developer's annotations

the cloudlet run the Kimberley Control Manager (KCM) to support the transient binding between themselves using a TCP tunnel established between these two KCMs.

The authors used VM overlay sizes and the speed of the synthesis process to evaluate the system performance. The VM overlay sizes were 100-200 MB for a collection of Linux applications. These sizes were an order of magnitude smaller than the full VM size (8 GB). The speed of synthesis ranged from 60 to 90 seconds. These results are acceptable for an unoptimized proof-of-concept prototype, and there is plenty of room for improvement through further optimization. For instance, a high-bandwidth short-range wireless network can reduce overlay transmission time, parallelism on the cloudlet can decrease decompression and overlay application times; caching as well as prefetching can be used to eliminate VM synthesis delays. The deployment challenges are also discussed, including 1) the business model (bottom-up versus top-down), 2) the sizing of cloudlets, i.e., how much processing power and storage capacity a cloudlet should provide, and 3) trust and security.

2. **Remote Execution in MAUI:** MAUI (Cuervo et al., 2010) was originally motivated by the assumption that battery technology will be a major bottleneck for the future growth of smartphones. MAUI consists of three main components. First, program partitioning uses the Microsoft .NET Common Language Runtime (CLR) to enable developers to annotate methods that may be performed remotely, to extract methods that may be performed remotely using reflection (Richter, 2010), and to identify the state of the application using type-safety and reflection. MAUI generates two proxies on both the mobile device and the server that handle control and data transfer to implement decisions on which methods to run remotely and which to run locally. Second, the MAUI profiler and solver will characterize the device and the program, then determine the methods to be executed remotely. On the server side, there is a MAUI coordinator handling the authentications and resource allocations.

The mobile part of MAUI was implemented on an HTC Fuze smartphone running Windows Mobile 6.5 with the .NET Compact Framework v3.5, and the MAUI server was implemented on a desktop with a dual-core 3 GHz CPU and 4 GB RAM running Windows 7 with the .NET Framework v3.5. The main results measure energy consumption and execution time for three applications—face recognition, 400 frames of a video game, and 30 moves in a chess game. The results show that using remote execution on MAUI saves 5-12 times the energy compared to the smartphone only case and reduces the execution time by more than a factor of 6.

3. **CloneCloud Utilizing Dalvik VM:** CloneCloud (Chun & Maniatis, 2009; Chun, Ihm, Maniatis, Naik, & Patti, 2011) allows a smartphone to partially offload its application to the phone's clone in the cloud. It migrates a modified version of the original application executable to a virtual machine in the cloud. This algorithm allows thread granularity migration, and therefore the User Interface (UI) or other essential components can remain to be executed at the mobile. Additionally, native methods can execute at both the mobile device and its clones in the cloud/cloudlet. One drawback of the CloneCloud approach is that local threads need to block unless they are independent from the migrated threads.

Chun *et al.* developed a dynamic profiler to analyze the execution time and energy cost of each method on a mobile device, which are then used by an optimization solver to decide which method(s) should be migrated to the clone. The profiler and

optimization solver were implemented on a modified Dalvik VM on Android, and this requirement may limit the scope of its application. CloneCloud is tested on an unlocked HTC G1 Android phone and a server with a 3.0 GHz Xeon CPU running the Android x86 virtual machine via VMware ESX 4.1. Three applications—a virus scanner, image search, and privacy-preserving targeted advertising—were tested on the CloneCloud prototype. The results show that for the tested applications, when connecting to the CloneCloud via Wi-Fi, the execution time is shortened by 2.1x-20x and the energy consumption is reduced by 1.7x-20x. When connecting to the CloneCloud via 3G, the execution time is shortened by 1.2x-16x and the energy consumption is reduced by 0.8x-14x.

4. **Virtual Smartphone Over IP Utilizing Dalvik VM:** Chen *et al.* (Chen & Itoh, 2010; Chen, et al., 2012) introduce a framework that allows heavy backend tasks on an Android phone to be offloaded to an Android virtual machine in the cloud. Unlike MAUI or CloneCloud, the authors built an Android OS on an x86 cloud server on which a virtual smartphone is executed. Two frameworks are proposed: the first framework (Chen & Itoh, 2010) offloads an entire application to the virtual smartphone and controls the application through remote desktop sharing; the second one (Chen, et al., 2012) offloads only the compute-intensive components to the cloud. The former offers heightened security and data leakage prevention as the entire application and resulting data do not physically reside on the mobile, while the latter offers fast GUI responsiveness and offline execution.

The major advantages of using this approach over MAUI and CloneCloud are 1) no use of additional APIs in the source code is required, and 2) no modifications to the mobile device's OS or root access are required. Note that these features are useful for system deployment. To achieve the above features, the authors replace the AIDL (Android Interface Definition Language) tool with a helper tool so that the compiler automatically creates service wrappers that are offloaded to the cloud by a service offloader. Offloading decisions may be made according to the time and energy consumption required to perform a task. Once offloading is done, the user needs to wait until the task is completed before the service offloader re-evaluates the time and energy metrics.

5. **OSGi Approach:** In (Verbelen et al., 2012), Verbelen *et al.* introduced a different definition of a cloudlet. In their cloudlet architecture, the unit of deployment is a component. Components are managed by an Execution Environment that runs on top of an Operating System (OS). The OS is installed on a node that is either virtualized or real hardware, and managed by a Node Agent. A cloudlet is a group of nodes (either mobile devices and PCs or elastic cloud servers) that are physically proximate to each other. A Cloudlet Agent optimizes the performance by deploying or configuring the components within the cloudlet.

The proposed cloudlet framework is implemented on top of the OSGi framework (OSGi, 2012), allowing components to be installed, started, stopped, updated, and uninstalled without a reboot. The authors use an OSGi bundle named R-OSGi (Rellermeyer et al., 2007) to facilitate the distribution of components across different OSGi instances. In other words, the R-OSGi allows components to be executed on different platforms. The authors implement an augmented reality application to evaluate the cloudlet framework. Results show that with components being offloaded to a local laptop computer, the application on the mobile device can be improved to satisfy the

performance requirements. The experiment results also show that when the cloudlet is running in a distant cloud, the performance decreases to an unsatisfactory level due to the increased latency.

Other Middleware Designs

The above describes approaches that allow mobile devices to offload computational tasks to the cloudlet/cloud. In general, these approaches can be categorized as middleware that lies on top of the operating system and provide services to the applications. More specifically, the offloading approaches described above enable the communication and management of data and code between the client and the cloudlet/cloud. Besides supporting code migration, a middleware framework may also provide generic interfaces to handle the communication and input/output functions, which will facilitate the design of software for these mobile-cloud and mobile-cloudlet-cloud architectures.

One example of such a middleware framework design is given by Flores *et al.* (2011). In their paper, a generic middleware framework named Mobile Cloud Middleware (MCM) is introduced. MCM enables interoperability between the mobile and the cloud/cloudlet. In MCM, a mobile application first sends an HTTP or XMPP request to MCM, which processes this request and forwards the request to the MCM manager. An interoperability API engine within the manager then decides which API set to use to interact with the cloud/cloudlet. When the process running on the cloud/cloudlet is finished, a notification is sent to the mobile device using the push notification services (C2DM–Cloud to Device Messaging (C2DMF, 2012) for the Android platform and APNS–Apple Push Notification Service (Apple, 2012) for the iOS platform). This request-notification mechanism is processed asynchronously, so that the mobile device can perform other tasks while waiting for the notification.

For some applications, it is important for the cloud/cloudlet to have the capabilities to dynamically capture and utilize contextual information from mobile devices to improve QoS. Such contextual information may involve user profiles, session quality, network conditions and environmental conditions such as temperature, humidity, and location. In Hoang and Chen (2010), Hoang et al. summarize the functions that context-aware middleware is expected to incorporate, including 1) intelligent monitored data analysis that preprocesses raw sensor data to improve its quality and to update context repositories, 2) network auto-switch that monitors network latency and automatically chooses the best network, and 3) energy consumption management that aims to minimize energy consumption at the device level, the communication level, or the collaborative level. This middleware layer constructs a communication bridge between the data acquisition layers on both the mobile and the cloud service ends.

Task Distribution Algorithms

With the support of middleware, the mobile devices are able to offload their computationally-intensive application components to one or multiple of the resource-rich cloudlets or cloud servers. In order to fully maximize the benefits of utilizing the cloud resources, task distribution algorithms must be developed.

In the MAUI approach, a MAUI profiler is used to estimate the characteristics of the device's energy consumption, the program's runtime and the resource needs, as well as the characteristics of the wireless network such as bandwidth, packet loss rates and delay. Then, the MAUI solver determines which methods can be remotely executed based on the information computed by the MAUI profiler. The solver uses Integer Linear Programming (ILP) to solve an optimization problem whose objective function is to maximize the energy savings given constraints about latency penalty and methods

that may be computed remotely. A similar profiler and a similar solver were used by CloneCloud to determine the migration point.

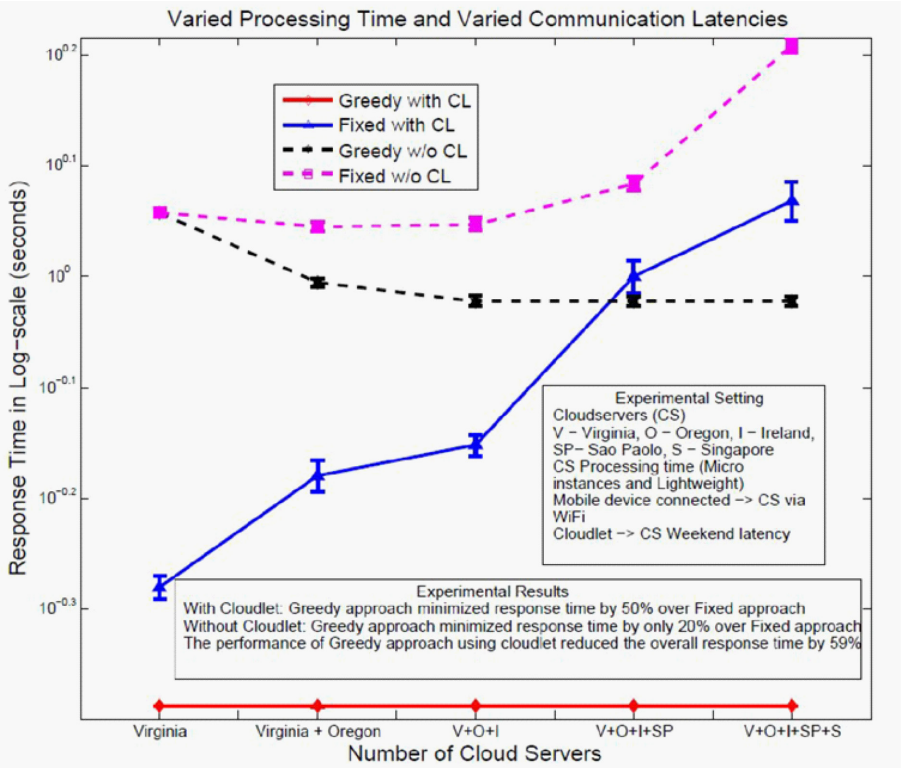
When the network connectivity is intermittent, extra latency will be introduced if the optimization algorithm uses the current communication condition to determine the migration point, such as being used in CloneCloud. In Cirrus Cloud (Shi et al., 2012), Shi *et al.* introduce an offloading algorithm that recursively chooses the optimal migration points from the root of the profile tree of an application. At every node within the tree, the algorithm computes the completion time to decide whether to execute the entire subtree locally, migrate it to the cloud entirely or migrate only parts of it. When migrating parts of the subtree, the same algorithm is iteratively applied to all the children of this node. With the computation and future network connectivity accurately known, the algorithm is able to find the optimal partitioning of the application and minimize the execution time. In reality, it is obvious that future network connectivity is not known ahead of time. However, using historical statistics may help to predict the connectivity and therefore achieve a close-to-optimal code migration.

The above approaches consider the code offloading from one mobile client to one server. In the cases when multiple servers may be used, the latency of each individual server must be considered. As indicated by Table 2, the response latencies of different cloud servers have significant variations. This diversity of connectivity creates the potential for gains through the smart selection of cloud servers for the offloading of computation. Motivated by this potential, the authors of MOCHA developed two task distribution algorithms, namely the fixed and the greedy algorithms, to optimize the mobile-cloud computing performance in terms of result response time.

The fixed algorithm is used to evenly distribute the pending tasks to the cloud servers (and the cloudlet if there is one). On the other hand, the greedy algorithm continuously sends the next pending task to the server (or cloudlet) that is able to return the result in a minimum amount of time. This process is repeated until all the pending tasks are assigned. The authors conduct Monte-Carlo simulations to analyze the effects of using the fixed algorithm and the greedy algorithm on a mobile-cloud network with a cloudlet or without a cloudlet. In the simulations, a computational job consisting of 5 identical and independent tasks is distributed among available cloud servers (and a cloudlet) with varied processing capabilities and communication latencies. As shown in Figure 3, the greedy algorithm reduces the overall response time by 50% over the fixed approach using the cloudlet, while only a 20% gain is achieved without the cloudlet. The results demonstrate the benefits of using the greedy task distribution algorithm as well as the benefits of using the MOCHA architecture.

While the above approaches all try to maximize the performance of the mobile device side, in (Hoang, Niyato, & Wang, 2012), Hoang *et al.* introduce an admission control policy that stands on the cloud server's side. The authors propose an optimization model based on a semi-Markov decision process to maximize the reward (e.g., revenue of service provider) of the resource usage in the cloudlet under resource and bandwidth constraints while meeting the QoS requirements (i.e., mobile users' service requests accept rate). The optimization model is transformed into a linear programming model and can be easily solved by a standard linear program solver. In the paper, the authors consider that the offloaded application partitions will be processed in the cloudlet rather than forwarded to the cloud. Therefore, the bandwidth and resource limitations at the cloud are not included in the model. According to the model, the control policy decides whether the service request from a user should be accepted or blocked. In the performance evaluations, the authors assume a circumstance where two classes of users, i.e., members with higher priority and non-members with low priority, are using the

Figure 3. Simulated response times using a varied number of cloud servers



cloudlet. Two services with different bandwidth and resource requirements are considered. The results show that using the proposed control policy, under the bandwidth and resource constraints, the cloudlet is able to satisfy the members' QoS requirements while maintaining high resource utilization rate.

FUTURE RESEARCH DIRECTIONS

In this chapter, we provided an extensive survey of the state-of-the-art mobile-cloud computing techniques, some of which utilize cloudlets as the middle layer. We provided a summary of the existing architectural designs and compared different approaches that enhance application performance via cloud-based execution. We also highlighted the research and technological challenges in different approaches presented in the literature. While much work has been done to date, mobile-cloud computing is still in its early research stages. Especially, the cloudlet is a new topic in the cloud computing world. Before these mostly theoretical proposals for the cloudlet find their place in practical applications, many research challenges have to be overcome. In this section, we summarize the most important challenges.

Cloudlet Design

Two main components of the cloudlet are its hardware architecture and software management mechanism. In the literature, many envision to extend a Wi-Fi access point into a more intelligent machine equipped with cloudlet functionalities (Satyanarayanan et al., 2009; Soyata et al., 2012a, 2012b; Ha et al., 2012; Fesehaye et al., 2012). To support this vision, a determination must be made as to what is a reasonable amount

of compute power and storage capacity that can be incorporated into the cloudlet without exceeding the power consumption and equipment cost constraints. For example, a cloudlet that costs as much as a desktop PC and consumes as much power is unlikely to be adopted by the masses. Alternatively, a cloudlet that does not have sufficient compute power will not augment the mobile devices' capabilities enough to make an impact on the overall performance. Therefore, ideal cloudlet architecture parameters lie between these two extremes. Such questions are closely related to the deployment strategy that centers on the business model with incentives.

The primary questions regarding software are 1) support for a variety of applications, 2) self-managing environments, and 3) efficient resource management. The system software environment should be generic enough so that different kinds of applications can execute without major modifications; the cloudlet resources should be managed automatically with minimal human involvement; and the resource (processing, storage, and networks) usage should be optimized so that cloudlet computation can support as many applications as possible at a given time and the overall execution time can be minimized. We envision cloudlets incorporating modern processors, such as GPUs (GeForce500, 2011), and modern memory subsystems with potentially specialized memory-based accelerators (Soyata & Liobe, 2012; Guo et al., 2010).

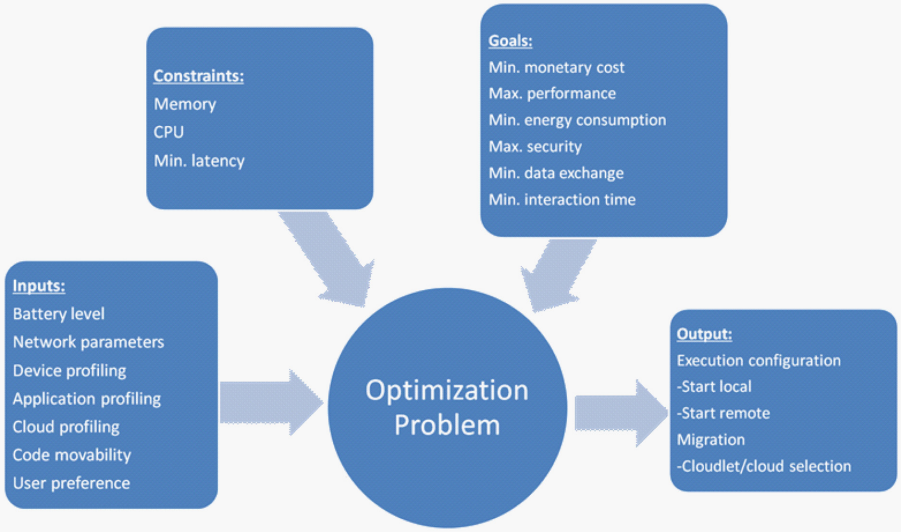
Task Distribution

Current implementations such as MAUI (Cuervo et al., 2010) and CloneCloud (Chun & Maniatis, 2009) have utilized offloading algorithms, where the code in the mobile device runs in a Virtual Machine (VM) and the execution can be migrated between the mobile and the cloud in real-time. However, there still remain many techniques that can be explored for further performance improvements by migrating the execution across multiple cloud servers, pipelining the transmission of application partitions to hide the transmission delay, and caching the reusable partitions to reduce the transmission load. As discussed previously, multiple cloud service providers or ad hoc cloud servers may be employed for computational or storage resources. This increases the complexity of the task distribution problem. Although the authors of MOCHA consider the computation power and network latency of different cloud servers when assigning the tasks (Soyata et al., 2012a, 2012b), it is necessary to develop more generic task distribution algorithms that take into consideration the resources and the constraints of the mobile devices, cloudlets and the cloud servers. Using a more comprehensive cost model, such as the one shown in Figure 4, is needed to develop better dynamic optimization algorithms to further enhance the performance and robustness. With sufficient computation power, a cloudlet is a proper candidate to optimize the task distribution decision dynamically.

Security and Privacy

As many mobile devices and the cloudlet/cloud collaborate and share data, security and privacy is always an important issue. While WPA2 (Wi-FiAlliance, 2012) and IPsec (BBN, 2005) provide layer-2 encryption of the data, layer-6 encryption is still a requirement for some applications. For example, layer-6 encryption is critical for pharmaceutical applications such as those involving bioinformatics or computational chemistry that are executed remotely on rented/commercial cloud platforms (AWS, 2012; Microsoft, 2012; Google, 2012). Homomorphic encryption can allow the computation to be performed without ever decrypting the data, providing additional layers of security. Future work is required to determine

Figure 4. The cost model of mobile cloud computing (adapted from Zhang, Kunjithapatham, Jeong, & Gibbs, 2011; Kovachev, Cao, & Klamma, 2011, and reprinted with permission of the authors)



how layer-6 encryption, including homomorphic encryption, can be applied when passing data between the mobile, cloudlet and cloud.

Energy Efficiency

As more hand-held mobile devices are equipped with sensing capabilities, collaborative sensing applications have become a reality. These applications often require thousands of participating smartphones that do opportunistic sensing with little user involvement. Since this opportunistic sensing may deplete the battery rather rapidly, it is crucial to implement effective resource management strategies to maximize the battery life of these phones. We should consider interactions between mobiles and the cloud as well since heavy communications consume large amount of battery power. We can model this as an optimization problem for optimal resource management and compute the best strategy for a given network topology, battery power, and network conditions.

Support for Mobile Application Developers

Developer tools such as software libraries with clearly defined APIs will increase the development productivity of mobile-cloud computing systems. The libraries will also help improve the system performance, efficiency, and compatibility while reducing the chances of faulty design and implementation. These APIs and libraries should be easily extensible, easy-to-use, and transparent to users so that users do not have to have knowledge about implementation details.

ACKNOWLEDGMENT

This research was funded in part by UCB Pharma and by CEIS, an Empire State Development-designated Center for Advanced Technology. The authors thank NVIDIA Corporation for their support of our research and Hemang Thakkar for his support in gathering the communication latency measurement data reported in this chapter.

REFERENCES

AES. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Ali, M. (2009). Green cloud on the horizon. *Cloud Computing*, 451-459.

Anderson, D. P. (2004). BOINC: A system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004* (pp. 4-10). IEEE.

Android. (2012). Retrieved from http://www.android.com/

Apple. (2012). *Apple push notification service.* Retrieved from http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html

AWS. (2012). *Amazone web services.* Retrieved from http://aws.amazon.com

Ba, H., Heinzelman, W., Janssen, C., & Shi, J. (2013). Mobile computing - A green computing resource. In Proceedings of the Wireless Communications and Networking Conference (WCNC) (pp. 4451-4456). IEEE.

BBN. (2005).*Security architecture for the internet protocol.* Retrieved from http://tools.ietf.org/pdf/rfc4301.pdf

BOINC. (2012). *Native BOINC for Android.* Retrieved from http://nativeboinc.org/site/uncat/start

C2DMF. (2012). *Android cloud to device messaging framework*. Retrieved from https://developers.google.com/android/c2dm/

Chen, E., Ogata, S., & Horikawa, K. (2012). Offloading Android applications to the cloud without customizing Android. In *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),* (pp. 788--793). IEEE.

Chen, E. Y., & Itoh, M. (2010). Virtual smartphone over IP. In *Proceedings of the 2010 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM),* (pp. 1-6). IEEE.

Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (pp. 301-314). IEEE.

Chun, B. G., & Maniatis, P. (2009). Augmented smartphone applications through clone cloud execution. In *Procedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS).* HotOS.

Corson, M. S., Laroia, R., Li, J., Park, V., Richardson, T., & Tsirtsis, G. (2010). Toward proximity-aware internetworking. *IEEE Wireless Communications*, *17*(6), 26–33. doi:10.1109/MWC.2010.5675775.

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (pp. 49-62). ACM.

Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2011). *A survey of mobile cloud computing: Architecture, applications, and approaches*. Wireless Communications and Mobile Computing. doi:10.1002/wcm.1203.

DOCSIS. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/DOCSIS

Eastlack, J. R. (2011). *Extending volunteer computing to mobile devices*. (Master's thesis). Las Cruces, New Mexico: New Mexico State University.

Fernando, N., Loke, S. W., & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 84–106. doi:10.1016/j.future.2012.05.023.

Fesehaye, D., Gao, Y., Nahrstedt, K., & Wang, G. (2012). Impact of cloudlets on interactive mobile cloud applications. In *Proceedings of Enterprise Distributed Object Computing Conference (EDOC)* (pp. 123-132). IEEE.

Flores, H., Srirama, S. N., & Paniagua, C. (2011). A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia* (pp. 87-94). ACM.

GeForce500. (2011). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/GeForce_500_Series

GeForce600. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/GeForce_600_Series

Google. (2012). *Google app engine.* Retrieved from http://code.google.com/appengine

Guo, X., Ipek, E., & Soyata, T. (2010). Resistive computation: avoiding the power wall with low-leakage, STT-MRAM based computing. In *ACM SIGARCH Computer Architecture News* (pp. 371–382). ACM.

Ha, K., Pillai, P., Lewis, G., Simanta, S., Clinch, S., Davies, N., & Satyanarayanan, M. (2012). *The impact of multimedia applications on data center consolidation*. Pittsburgh, PA: Carnegie Mellon University, School of Computer Seience.

HIPAA. (1996). Retrieved from http://www.hhs.gov/ocr/privacy/index.html

Hoang, D. B., & Chen, L. (2010). Mobile cloud for assistive healthcare (MoCAsH). In *Proceedings of the Asia-Pacific Services Computing Conference (APSCC)* (pp. 325-332). IEEE.

Hoang, D. T., Niyato, D., & Wang, P. (2012). Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In *Proceedings of the Wireless Communications and Networking Conference (WCNC)* (pp. 3145-3149). IEEE.

Intel. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Intel_Tick-Tock

IOT. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Internet_of_Things

Kovachev, D., Cao, Y., & Klamma, R. (2011). Mobile cloud computing: a comparison of application models. *arXiv preprint arXiv:1107.4940.*

Kumar, K., Liu, J., Lu, Y.-H., & Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, *18*(1), 129–140. doi:10.1007/s11036-012-0368-0.

Marinelli, E. (2009). *Hyrax: Cloud computing on mobile devices using mapreduce*. (Master's Thesis). Carnegie-Mellon University, Pittsburgh, PA.

Microsoft. (2012). *Windows azure.* Retrieved from http://www.microsoft.com/windowazure

NIST. (2001). Retrieved from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

OSGi. (2012). Retrieved from http://www.osgi.org/

Pattichis, C. S., Kyriacou, E., Voskarides, S., Pattichis, M. S., Istepanian, R., & Schizas, C. N. (2002). Wireless telemedicine systems: An overview. *Antennas and Propagation Magazine*, *44*(2), 143–153. doi:10.1109/MAP.2002.1003651.

PIC32. (2012). *Microchip*. Retrieved from http://www.microchip.com/pagehandler/en-us/family/32bit/

Qualcomm. (2012). Retrieved from http://www.qualcomm.com/snapdragon

Rellermeyer, J. S., Alonso, G., & Roscoe, T. (2007). R-OSGi: Distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware* (pp. 1-20). New York: Springer-Verlag.

Richter, J. (2010). *CLR via c*. Microsoft Press.

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing / IEEE Computer Society [and] IEEE Communications Society*, *8*(4), 14–23. doi:10.1109/MPRV.2009.82.

Shi, C., Ammar, M. H., Zegura, E. W., & Naik, M. (2012). Computing in cirrus clouds: The challenge of intermittent connectivity. In *Proceedings of the MCC Workshop on Mobile Cloud Computing* (pp. 23-28). ACM.

Soyata, T. (1999). *Incorporating circuit level information into the retiming process*. (Ph.D. thesis). Rochester, NY: University of Rochester.

Soyata, T., & Friedman, E. G. (1994). Synchronous performance and reliability improvement in pipelined ASICs. In *Proceedings of the Seventh Annual IEEE International ASIC Conference and Exhibit,* (vol. 3, pp. 383-390). IEEE.

Soyata, T., Friedman, E. G., & Mulligan, J. H., Jr. (1993). Integration of clock skew and register delays into a retiming algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems,* (pp. 1483-1486). IEEE.

Soyata, T., Friedman, E. G., & Mulligan, J. H., Jr. (1995). Monotonicity constraints on path delays for efficient retiming with localized clock skew and variable register delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems,* (pp. 1748--1751). IEEE.

Soyata, T., & Liobe, J. (2012). pbCAM: Probabilistically-banked content addressable memory. In *Proceedings of the IEEE International System-on-Chip Conference* (pp. 27-32). Niagara Falls, NY: IEEE.

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., & Heinzelman, W. (2012). Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Proceedings of the Symposium on Computers and Communications (ISCC)* (pp. 59-66). IEEE.

Soyata, T., Muraleedharan, R., Langdon, J., Funai, C., Ames, S., Kwon, M., & Heinzelman, W. (2012). COMBAT: Mobile-cloud-based compute/communications infrastructure for battlefield applications. In *Proceedings of SPIE Defense, Security, and Sensing* (pp. 84030K-84030K). International Society for Optics and Photonics.

Tegra3. (2012). *NVIDIA*. Retrieved from http://www.nvidia.com/object/tegra-3-processor.html

Tegra. (2012). *Wikipedia.* Retrieved from http://en.wikipedia.org/wiki/Tegra

Varshney, U. (2007). Pervasive healthcare and wireless health monitoring. *Mobile Networks and Applications*, *12*(2-3), 113–127. doi:10.1007/s11036-007-0017-1.

Verbelen, T., Simoens, P., De Turck, F., & Dhoedt, B. (2012). Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services* (pp. 29-36). ACM.

WiFiAlliance. (2012). Retrieved from http://www.wi-fi.org/knowledge-center/glossary/wpa2%E2%84%A2

Wood, A., Stankovic, J., Virone, G., Selavo, L., He, Z., & Cao, Q. et al. (2008). Context-aware wireless sensor networks for assisted living and residential monitoring. *IEEE Network*, *22*(4), 26–33. doi:10.1109/MNET.2008.4579768.

Zhang, X., Kunjithapatham, A., Jeong, S., & Gibbs, S. (2011). Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 270–284. doi:10.1007/s11036-011-0305-7.

KEY TERMS AND DEFINITIONS

Cloud: The platform of multiple servers over a widely disbursed geographic area, connected by the Internet for the purpose of serving data or computation.

Cloudlet: The intermediate device located between the mobile and the cloud to accelerate-mobile-cloud computing.

Computation Offloading: The process of a computational device (e.g., mobile) sending a given task to a different computational device (e.g., cloudlet).

Graphics Processing Unit (GPU): An accelerator device typically plugged into the PCI express bus of a computer to accelerate graphics and other massively parallel computations.

Mobile-Cloud Computing: The co-execution of a mobile application within the expanded mobile/cloud computational platforms to optimize an objective function.

Mobile Computing: The ability to use mobile devices to perform computing tasks without being limited to pre-defined geographical locations.

Smartphone: A mobile phone that has advanced computing capabilities and is built on a mobile operating system capable of running third-party applications.

Task Partitioning Algorithm: An algorithm that determines how to optimally execute a large task by executing its different subtasks at existing computational resources, e.g., mobile, cloudlet, and the cloud.