Applied Computing and Informatics (2016) xxx, xxx–xxx

Saudi Computer Society, King Saud University

Applied Computing and Informatics

(http://computer.org.sa)
www.ksu.edu.sa
www.sciencedirect.com

Mobile cloud computing for computation offloading: Issues and challenges

Khadija Akherfi [a,*], **Micheal Gerndt** [a], **Hamid Harroud** [b]

[a] *Technical University of Munich, TUM, Munich, Germany*
[b] *Al Akhawayn University in Ifrane, Ifrane, Morocco*

Received 28 September 2016; revised 3 November 2016; accepted 28 November 2016

KEYWORDS

Cloud computing;
Mobile cloud computing;
Computational offloading;
Algorithms;
Partitioning

Abstract Despite the evolution and enhancements that mobile devices have experienced, they are still considered as limited computing devices. Today, users become more demanding and expect to execute computational intensive applications on their smartphone devices. Therefore, Mobile Cloud Computing (MCC) integrates mobile computing and Cloud Computing (CC) in order to extend capabilities of mobile devices using offloading techniques. Computation offloading tackles limitations of Smart Mobile Devices (SMDs) such as limited battery lifetime, limited processing capabilities, and limited storage capacity by offloading the execution and workload to other rich systems with better performance and resources. This paper presents the current offloading frameworks, computation offloading techniques, and analyzes them along with their main critical issues. In addition, it explores different important parameters based on which the frameworks are implemented such as offloading method and level of partitioning. Finally, it summarizes the issues in offloading frameworks in the MCC domain that requires further research.

Contents

* Corresponding author.
E-mail addresses: k.akherfi@aui.ma (K. Akherfi), gerndt@in.tum.de (M. Gerndt), h.harroud@aui.ma (H. Harroud).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.aci.2016.11.002

1. Introduction

The main goal of CC is to allow IT departments to focus on their businesses and projects instead of just taking care of their data centers and keeping them working [2,18,20]. CC is a new concept that aims to provide computational resources as services in a quick manner, on demand, and paying as per usage. The CC paradigm is presented in three cloud delivery models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) as shown in Fig. 1. According to Gartner [3], CC will have in 2016 a Global Compounded Annual Growth Rate (CAGR) of IaaS: 41%, PaaS: 26.6% and SaaS: 17.4%.

Recently, user preferences for computing have changed because of the latest developments and enhancements in mobile computing technologies. Several reports and studies have presented the importance of MCC and its impact on mobile clients and enterprises. For instance, and according to a recent study by ABI Research, more than 240 million business will use cloud services through mobile devices by 2015 and this will push the revenue of the MCC to $5.2 billion [11]. Moreover, the usage of smartphones has increased rapidly in various domains, including enterprise, management of information systems, gaming, e-learning, entertainment, gaming, and health care. Although the predictions that mobile devices will be dominating the future computing devices, mobile devices along with their applications are still restricted by some limitations such as the battery life, processor potential, and the memory capacity of the SMDs [31]. Nowadays, modern mobile devices have sufficient resources such as fast processors, large memory, and sharp screens. However, it is still not enough to help with computing intensive tasks such as natural language processing, image recognition, and decision-making. Mobile devices provide less computational power comparing to server computers or regular desktops and computation-intensive tasks put heavy loads on battery power.

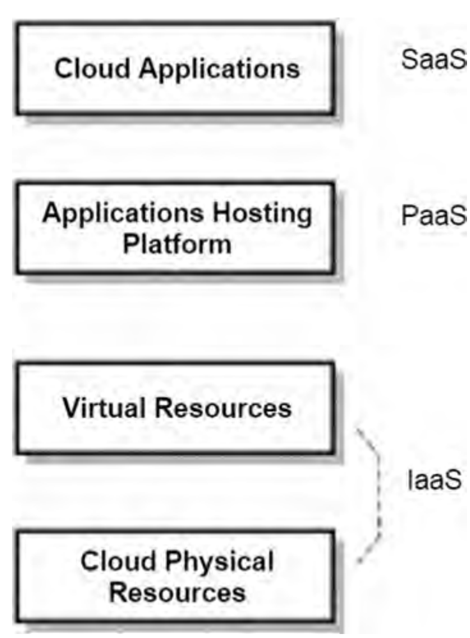

Figure 1 Cloud computing layers.

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

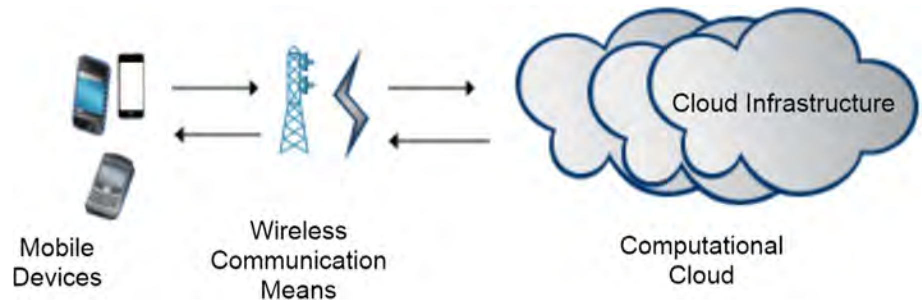

Figure 2 General view of MCC.

Currently, there are several works and research in CC that aim at enhancing the computing capabilities of resource-constrained mobile client devices by providing mobile clients access to cloud infrastructures, software, and computing services. For example, Amazon web services are used to protect and save clients' personal data via their Simple Storage Service (S3) [23]. In addition, there are several frameworks that allow to process data intensive tasks remotely on cloud servers. For instance, the ASM computation offloading framework [6] showed that computation offloading helped to reduce the energy consumption cost of mobile devices by 33%, and the turnaround time of the application by 45% [30].

The following two points highlight our main contribution in this paper.

1. Classifying current computation offloading frameworks. Analyzing them by identifying their approaches and crucial issues.
2. Presenting the related open issues in computation offloading for MCC and challenges that require more investigation and elaboration.

This paper is organized as follows: Section 2 explains the essential background concepts and terminology, including CC, the MCC concept, and computation offloading. Section 3 presents the offloading approaches. A comparison between the different offloading frameworks and their critical issues is discussed in Section 4. Section 5 highlights general issues and challenges in computation offloading for MCC. Finally, Section 6 gives a summary and points to future work.

2. Concepts and background

2.1. Cloud computing

CC is a new way of providing computing resources and services. It refers to an on-demand infrastructure that allows users to access computing resources anytime from anywhere [25]. CC offers to users and business three main advantages: (1) enormous computing resources available on demand, (2) payment for use as needed and on a short-term basis (storage by the day and release them as needed), and (3) simplified IT management and maintenance capabilities [1]. CC provides clients with different applications as services via the Internet. As examples of public CC we can list Windows Azure and Amazon Web Services (AWS). Windows Azure is an open and flexible cloud platform which provides several services to develop, deploy and run web applications and services in cloud data centers [33]. AWS, which is considered as an example of a public computing tool, provides users with two models: infrastructure as a service and software as a service. These services allow the user to use virtualized resources in cloud data centers [23]. Computational clouds implement a variety of service models in order to use them in different computing visions [4].

2.2. Mobile cloud computing

MCC can be seen as a bridge that fills the gap between the limited computing resources of SMDs and processing requirements of intensive applications on SMDs.

The Mobile Cloud Computing Forum defines MCC as follows [11]: "Mobile Cloud Computing at its simplest form refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers".

MCC has attracted the attention of business people as a beneficial and useful business solution that minimizes the development and execution costs of mobile applications, allowing mobile users to acquire latest technology conveniently on an on-demand basis.

Fig. 2 shows the general view of MCC which is composed of three main parts: the mobile device, wireless communication means, and a cloud infrastructure that contains data centers. These latter provide storage services, processing, and security mechanisms for both the cloud environment and mobile devices.

2.3. Computation offloading

Computation offloading is the task of sending computation intensive application components to a remote server. Recently, a number of computation offloading frameworks have been proposed with several approaches for applications on mobile devices. These applications are partitioned at different granularity levels and the components are sent (offloaded) to remote servers for remote execution in order to extend and enhance the SMD's capabilities. However, the computation offloading mechanisms are still facing several challenges.

In the remaining part of this section, our objective was to give a summary about the MCC offloading research by discussing the following:

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

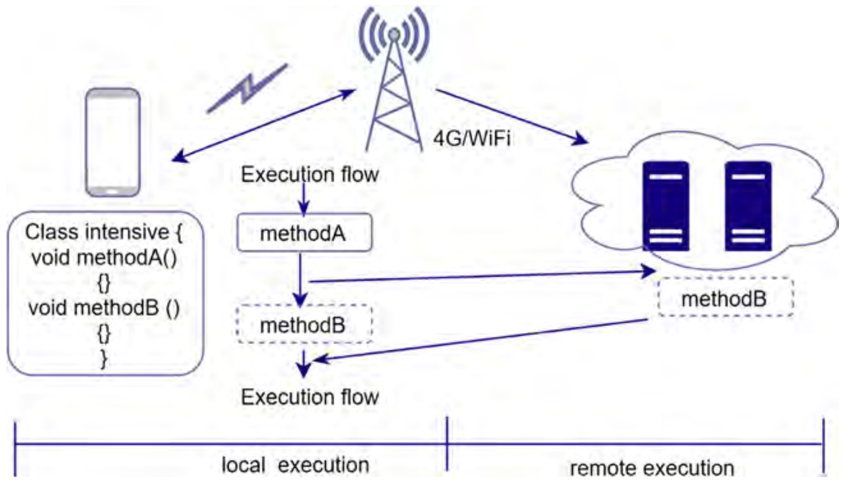

Figure 3 Offloading process overview.

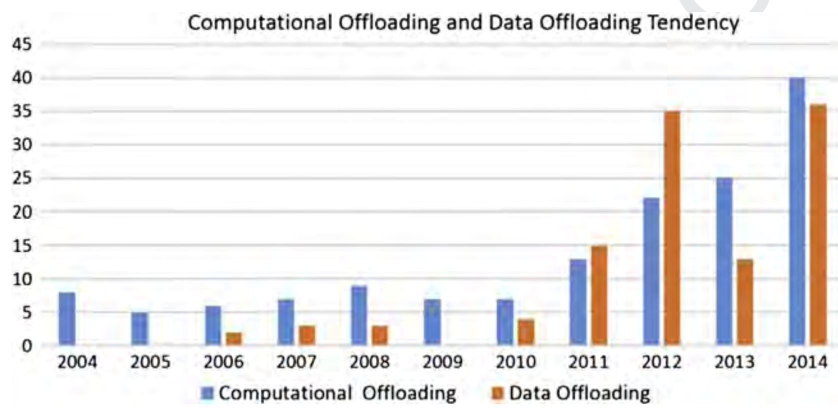

Figure 4 Number of computation offloading and data offloading papers.

1. Usage scenarios for offloading in MCC.
2. Techniques being applied in offloading.
3. A classification of proposed offloading frameworks.

Computation offloading emerged around 1970s. However, its potential has been widely explored only when wireless communication and Internet speed became sufficiently enhanced and could support it [37].

The potential of mobile offloading mainly depends on the mobile network technologies such as cellular and WiFi. They determine the viability of mobile offloading. Today, the WiFi technology is able to provide high bandwidth connections. However, the data transmission using the cellular network requires a considerable amount of energy from the mobile device as opposed to a WiFi network.

Fig. 3 illustrates the environment that supports computation offloading. In this overview, the mobile device decides to offload method B to a cloud server or a powerful machine. The cloud here provides the virtual computation resources to run the offloaded components. The powerful machine can be a server or cluster in a computing center, or a computing grid, or a virtual server in the cloud. Fig. 4[1] shows the number of published papers since 2004 citing the word "Offloading" and "Computation". Most of the research works tackling data offloading have the goal to store data in remote large repositories. It can be seen in Fig. 4 that the research work citing "computation offloading" and "data offloading" is increasing progressively.

Clearly, computation offloading is worthwhile only when the local execution (mobile device) consumes more time and energy than the offloading overhead. Many factors can impact the offloading decision and could influence the offloading process.

Fig. 5 illustrates these factors which are network specifications, mobile specifications, application characteristics, server specifications, and user's preferences.

The computation offloading has experienced a remarkable improvement which makes it applicable in a wide range of domains. Table 1 shows some of these domains.

As it is known, the battery life of mobile devices and the limited processors' capabilities remain key limiting factors in the design of mobile applications. Today, the demand for resource intensive applications such as 3D video games and voice recognition is increasing day by day. To close this gap between the users demand and the mobile devices limitations, research studies have been exploring computation offloading in MCC to bring the power of cloud computing to the otherwise limited mobile devices capacity.

[1] The number of published works is retrieved from Google Scholar.

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

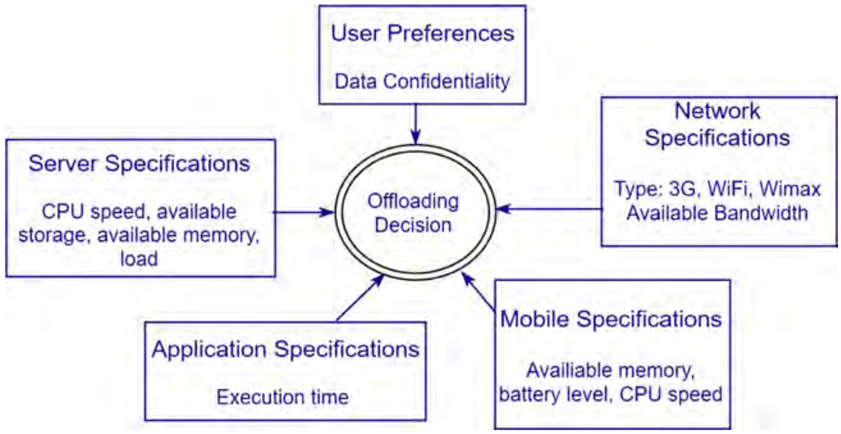

Figure 5 Aspects affecting the offloading decision.

Table 1 Areas of the research work related to computation offloading.

Domain	Research work	Contribution of some research work
Artificial intelligence based applications	[6,8,16,27,37]	The proposed work aims to reduce computation time on robots by using an offloading technique. The system is designed for real-time moving object recognition and tracing using computation offloading [27]
Graphics and image processing	[6,37,17,35]	The suggested work examines the trade-offs that emerge from executing some of the workload locally and some on remote cloud servers. Extraction and matching are two features that are crucial in image classification. The paper analyzes the possibility of executing the previously mentioned features in a mobile device using different scenarios [17]
Health and social applications	[24,21]	The proposed work presents an extensible module that proactively facilitates the management of processes for web service supported by mobile applications. This module measures power consumption and application performance from the smartphone device. Based on the obtained measurements, the module dispatches image processing jobs locally or remotely [24]
Games applications	[35]	The objective of the presented work was to satisfy Cloud Mobile Gaming communication and computation constraints by using an adaptation technique that ensures a good mobile gaming experience [35]
Mathematics	[7,34]	The work presents a new architecture that addresses the mobile device limitations by using a partial offloading execution from the smartphone to a remote cloud infrastructure hosting smartphone clones [7]
File system and database	[26,7]	The presented work addresses the maximization of the lifetime of mobile devices by developing computational offloading schemes while considering the network status. The paper presents an experimental environment where different profiles and computational moles are evaluated [26]

3. Offloading approaches

3.1. *Offloading steps*

Offloading transfers a compute intensive task from the SMD to a remote server. Offloading is a process that includes basically three steps: application partitioning, preparation, and offloading decision.

3.1.1. *Application partitioning*

The first step is application partitioning which is very important for computation offloading. It divides the application into offloadable and non-offloadable components meaning which components to retain on the mobile device and which to migrate to the cloud server. The decision whether a component is offloadable can be taken based on different information. The programmer can annotate application components for example through a special API as offloadable. Compute intensive parts that are candidates for offloading can be identified also by source code analysis in combination with performance prediction or via application profiling. If the partitioning is done at design time, both techniques have a limited accuracy since they do not take the real execution context into account, when the application is run.

3.1.2. *Preparation*

The preparation step performs all actions required for offloadable components to enable their use in mobile applications. This includes the selection of a remote server, the transfer and installation of the code, as the start of proxy processes that receive and execute tasks on behalf of the SMD. Besides the transfer of the code, also data might be transferred to prepare for the remote execution.

3.1.3. Offloading decision

The offloading decision is the final step before remote execution is started for offloadable components. Whether an installed remote component is used in the SMD application or not depends typically on the execution context. If the decision is taken at runtime, more precise information is available, for example, the SMD might even not have a wireless connection or the energy consumption for transferring the data for the remote execution might simply be too high. Whenever the situation changes, the offloading can be adapted. Such a runtime decision induces some overhead that typically is not present if the decision is taken at design time.

3.2. Framework classes

According to when the decision is taken to offload computation on a remote server, we can distinguish two types of offloading frameworks. The first class is static offloading frameworks. Here all the presented steps are performed at design time, before the application is started on the mobile device. The other classes are dynamic offloading frameworks. In those frameworks, at least the final decision whether to offload a computation is taken at runtime. The other two steps can be executed at design or runtime. For example, a framework that is based on user annotations of offloadable components and on pre-installation of the components on a remote server will be called a dynamic offloading framework, if it decides at runtime whether it is better to offload computation or not.

3.3. Framework mechanisms

Although there are several offloading mechanisms available for offloading computation intensive components of mobile applications to the cloud, we can classify these mechanisms into two broad categories:

1. Frameworks based on virtual machine cloning.
2. Frameworks based on code offloading.

Frameworks based on code offloading offload intensive application components by invoking a remote procedure call (RPC) using annotations, special compilation or binary modification. Whereas in virtual machine cloning, the mobile device's full image is captured and stored on the cloud server. During offloading, the mobile's execution is suspended and transferred to the VM clone in the cloud.

4. Comparison of offloading frameworks in mobile cloud computing

This section introduces different existing offloading frameworks. For each of the frameworks we identify the approaches used in the three steps introduced in the previous section. At the end of the section, the different frameworks will be compared with respect to their most important properties.

4.1. CloneCloud

Chun et al. present in [6] the CloneCloud framework which aims at improving the battery life and performance on the mobile device by offloading intensive components to cloud servers.

The partitioning step in this framework combines static program analysis with program profiling to produce a set of offloadable components while meeting some constraints, such as methods that use mobile sensors should be executed locally. The framework uses thread level granularity for partitioning of applications. The role of static analysis is to discover constraints on possible migration points while profiling aims to build a cost model for offloading and execution. Partitioning and integration of the applications are performed at the application level.

As a preparation step, a duplicate of the smartphone's software is captured and stored within a cloud server.

At runtime, the offloading decision is taken and threads are migrated from the mobile device to the clone in the cloud. In CloneCloud, threads must be suspended, all states of the threads must be transferred to the server, and then threads resume on the server in order to offload computation. The framework is based on VM instance migration to the cloud server. Fig. 6 shows the CloneCloud execution model. Initially, a duplicate of the smartphone's software is created in the cloud. The state of the smartphone and the clone is synchronized periodically or on-demand. After the execution of offloaded components, results from the execution on the clone are reintegrated back into the smartphone state. CloneCloud employs dynamic offloading.

The objective of the distributed execution mechanism in CloneCloud was to implement a specific partition of a given application process executing inside an application-layer virtual machine.

In CloneCloud framework, the distributed execution goes through several steps as follows. When the user tries to run a partitioned application, the framework looks in a database of pre-computed partitions for current execution conditions (such as available network bandwidth and cloud resources). The result of the verification is a partition configuration file. The partition is loaded by the application binary which instruments the selected methods with migration. On the mobile device, once the execution of the process reaches a migration point, the running thread is suspended and its state is wrapped and shipped to a synchronized clone. In this clone, the thread state is instantiated into a new thread with the same stack and

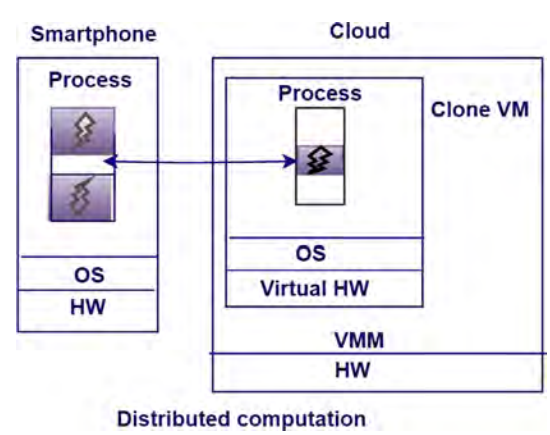

Figure 6 CloneCloud execution model (adapted from [6]).

reachable heap objects, and then resumed. On the cloud server, when the migrated thread reaches a re-integration point, it is suspended, packaged, and then shipped back to the mobile device. Finally, the received packaged thread is merged into the state of the original process in the mobile device.

To evaluate the prototype, the authors implemented three applications (virus scanner, image search, and privacy-preserving targeted advertising). All measurements are the average of five executions. Phone, CloneCloud with WiFi, and CloneCloud with 3G are the used environments.

A clear tendency is that larger workloads benefit from offloading because of amortization of the migration cost over a larger computation [6].

4.2. MAUI

MAUI [8] is a framework that considers energy saving on smartphones as the main objective function for the offloading process. MAUI is a highly dynamic offloading framework because of a continuous profiling process. The framework hides the complexity of a remote execution from the mobile user and gives the impression as if the entire application is being executed on the mobile device.

In MAUI, partitioning is done based on developer annotations to specify which components can be executed remotely and which cannot.

In the preparation step, two requirements should be met: (1) application binaries must be in both mobile and server sides and (2) proxies, profilers and solvers must be installed on both the mobile device and server sides.

At the beginning, MAUI profiler measures the device characteristics. Then, it keeps monitoring the program and network characteristics during the whole execution time because these characteristics can often change and any old or inaccurate measurement may lead MAUI to make the wrong decision.

The offloading decision is taken at runtime. The framework chooses which components should be remotely executed according to the decision of the MAUI solver. The decision is based upon the input of the MAUI profiler.

Fig. 7 shows the MAUI architecture. On the smartphone, the framework consists of the following components: a proxy, a profiler, and a solver. Each time a method is called, the MAUI profiler evaluates it for its energy-saving potential and profiles the device and the network to obtain the status information. Then, the MAUI solver works on the results provided by the profiler and determines the destination where the method will be remotely executed; the proxy is responsible for control and data transfer between the server and the smartphone. On the server side, the profiler and the server proxy perform similar roles as their client-side counterparts. The MAUI controller is responsible for authentication and resource allocation for incoming requests [8].

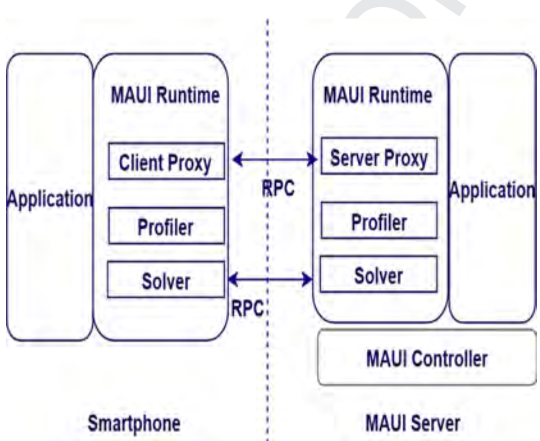

Figure 7 High-level view of MAUI's architecture (adapted from [8]).

The authors presents different experiments in order to compare the energy consumption of running three applications(-face recognition, chess, and video) standalone on the smartphone versus using MAUI for remote execution to servers that are located elsewhere. The face recognition application can achieve strong energy savings when using MAUI. On the one hand, the results of the conducted experiments showed that the energy consumed when offloading code using 3G is 2.5 times higher than offloading code to a close server. On the other hand, the energy savings for both video and chess game are less strong but they are still important; when offloading to a close server, MAUI saves 45% for chess and 27% energy for the video game.

4.3. Cloudlet

Offloading to the cloud is not always a solution, because of the high WAN latencies, mainly for applications with real-time restrictions. Thus the cloud has to be moved closer to the mobile user in the form of cloudlets.

Satyanarayanan et al. suggest in [29] a VM based cloudlet framework. A cloudlet can be defined as a hosting environment for offloaded tasks that is deployed to remote resources, as different as individual servers or parallel systems. Cloudlets are virtual-machine (VM) based on support scalability, mobility, and elasticity. They are located in single-hop nearness to mobile devices.

In the preparation step, the framework requires the cloning of the mobile device application processing environment to a remote host. It offloads the entire application using VM as the offloading mechanism and more precisely it uses a technique called dynamic VM synthesis. The VM would encapsulate and separate the guest software from the cloudlet's host software. The mobile device serves as a thin client providing only the user interface, whereas the actual application processing is performed on the cloudlet infrastructure.

Device mobility is the main critical issue for mobile users on the move while connected to cloudlets.

As Fig. 8 illustrates, cloudlets are widely distributed Internet infrastructure components whose storage resources and computing cycles can be exploited by nearby mobile devices while avoiding the long latency which is available for accessing distant cloud resources. These cloudlets would be situated in common areas, such as coffee shops, so that mobile devices can connect and work as a thin client.

Fig. 9 depicts the cloudlet architecture. Cloudlets are discoverable, and located in single-hop proximity of mobile devices. The main elements of the architecture are Cloudlet Host and Mobile Client. A Discovery Service is a component running in the cloudlet host and publishes cloudlet metadata. The cloudlet metadata (e.g. IP address and port to connect

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

Figure 8 Cloudlet illustration adapted from [29].

to the cloudlet) are used by the mobile client to specify the suitable cloudlet for computation offloading. Once the cloudlet is determined for offload, the mobile client sends the application code and the application metadata to the cloudlet server. The cloudlet server deploys the application code in the guest VM. Once the deployment is done, the execution of the application is launched.

We can take a scenario where the user must execute a computation intensive application. At runtime, the application discovers a nearby cloudlet and offloads the computation intensive mobile application [15]. However, because of loss of network connectivity, the mobile application can find a different cloudlet and run the application in a short time.

4.4. Jade

Sharing the same concern but from a different perspective, Qian et al. present in [28] a system that monitors application and device status and that automatically decides where the code should be executed. The goal of Jade was to maximize the benefits of energy-aware computation offloading for mobile applications while minimizing the burden on developers to build such an application.

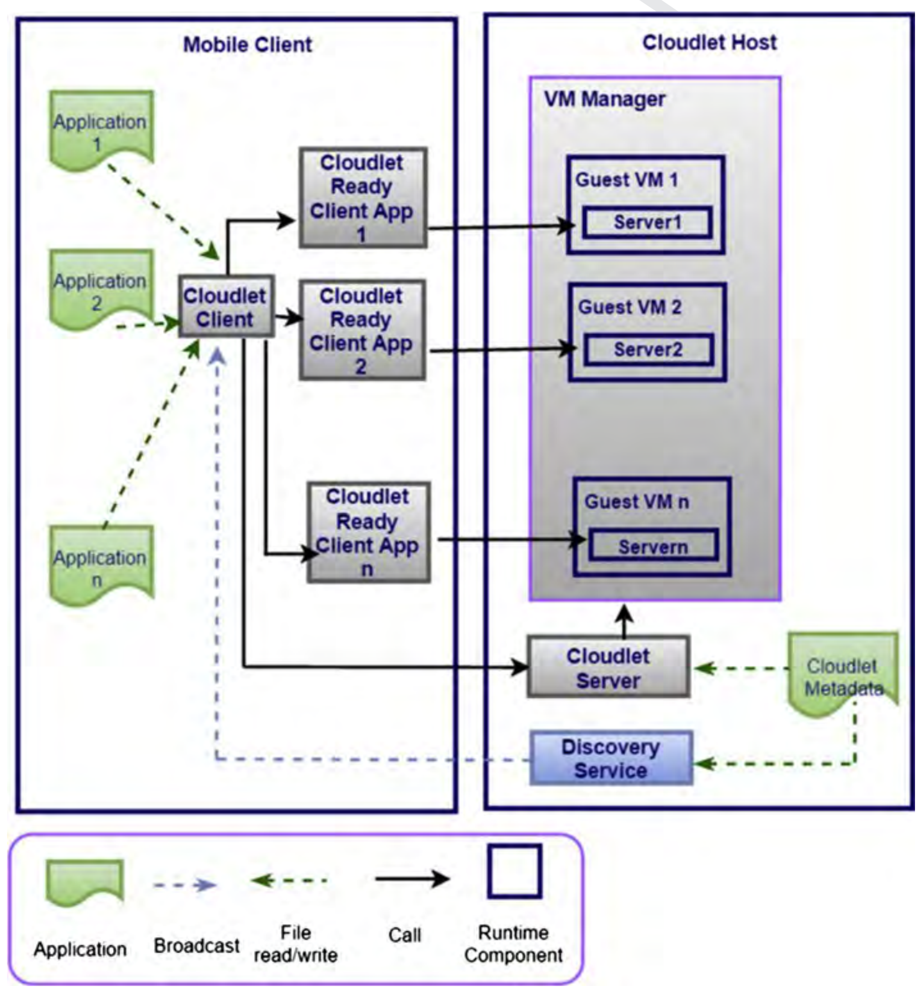

Figure 9 Cloudlet architecture (adapted from [22]).

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

During partitioning, applications are partitioned at the class level in Jade based on the collected information.

As a preparation, the system checks the application and device status by monitoring the communication costs, work load variation, and energy status. The framework provides a sophisticated programming model with a full set of APIs, so developers have total control on: how the application is partitioned, and how remote code interacts with local code.

The offloading decision is taken at runtime to decide where the code should be executed. Jade supports two types of servers: (1) Android servers and (2) Non-Android servers running operating systems such as Windows and Linux. Non-Android servers must have Java installed in order to support Jade. Jade's runtime engine runs as a Java program on a non-Android server.

Jade can dynamically change its offloading decision according to the device status and thus efficiently reduce energy consumption of mobile devices.

Fig. 10 presents an overview of the Jade framework. The mobile device that offloads computation is called the client. The device that executes the offloaded code is called the server. Mobile applications contain remote tasks which can be offloaded to the server. The Jade runtime engine automatically decides where to execute remote tasks and initiates distributed execution.

Fig. 11 presents the Jade framework architecture. In order to offload a computation, the system handles the following tasks:

- Profiling: In order to make correct offloading decisions, the framework should have updated information concerning the status of the application and the device. Application profiling is the process of collecting information about programs, such as energy consumption, data size, execution time, and memory usage. Similarly, device profiling collects information about devices status, such as battery level, CPU usage, and wireless connection.
- Communication: To offload code from the mobile client to the server, the system should be able to (1) connect to the other server; (2) coordinate with the remote server for offloaded components; (3) send data between the mobile device and the remote server; (4) follow status of remote execution; and (5) save information related to all connected devices (e.g., connection speed, hardware configuration).
- Optimization: The framework determines an optimized offloading approach to reduce energy consumption and enhance application's performance.

The Jade framework automatically transforms application execution on one mobile device into a remote execution optimized for wireless connection, power usage, and server capabilities.

In order to check the amount of energy that Jade can save for mobile device, authors run face detection application on a mobile device. The application performs face detection on 50 pictures with size of each less than 200 KB. Results showed that Jade reduces the power consumption for face detection application. Average power consumption was decreased by 34%.

4.5. Mirror server

Zhao et al. present in [38] the mirror server framework that uses Telecommunication Service Provider (TSP) based remote services. A TSP is a type of communication service provider which provides voice communication services such as landline telephone services. Mirror server extends capabilities of smartphones by providing three different types of services: computation offloading, security, and storage. Mirror server is a powerful server which retains VM templates for different mobile device platforms.

This framework does not require a partitioning as the entire application is offloaded.

In the preparation step, a new VM instance is created. This VM is called mobile mirror and the mirror server takes care of managing and deploying the mobile mirrors on a computing infrastructure in the telecom network. Applications are executed in the mirror VM instances and results are returned to the SMD. The framework employs an optimized mechanism for offloading.

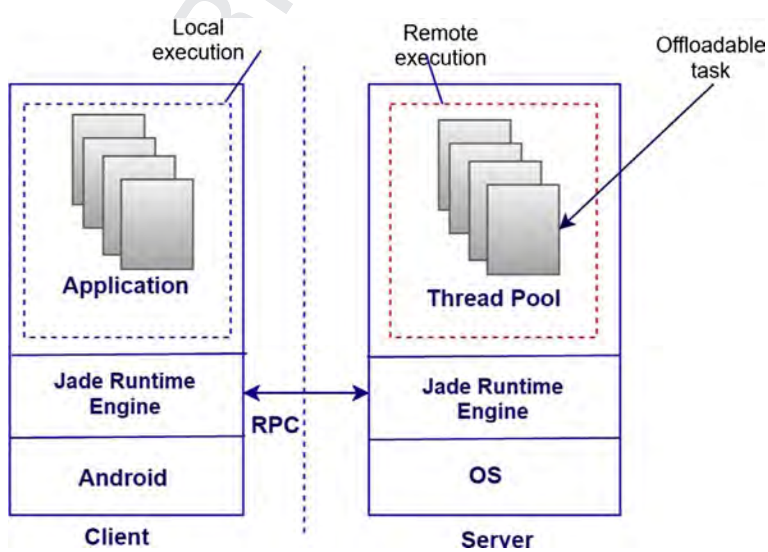

Figure 10 Jade overview (adapted from [29]).

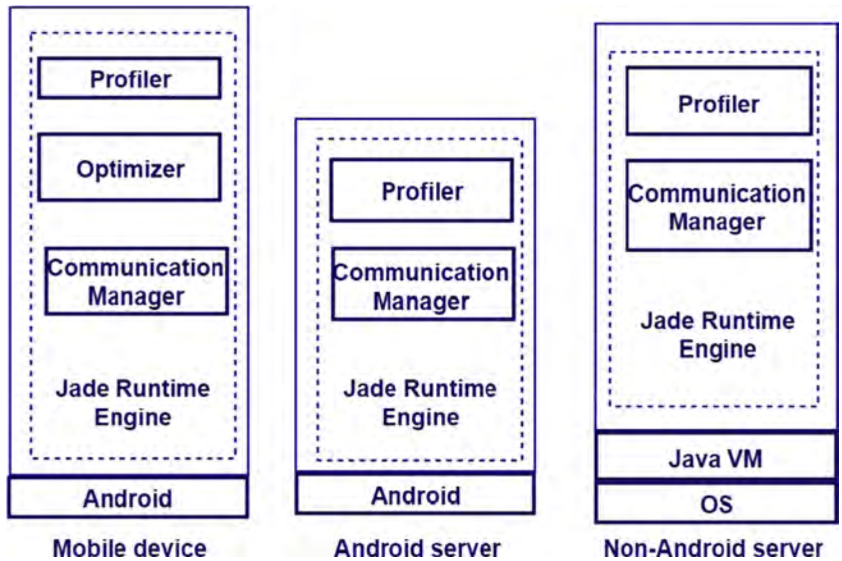

Figure 11 Jade architecture (adapted from [28]).

The Mirror Server architecture is presented in Fig. 12. On the SMD side, a client synchronization module (Syn-Client) is deployed within the SMD operating system (OS) to collect SMD input data and transmit them to the mirror server for synchronization. On the server side, in order to keep mirrors and smartphones synchronized, the synchronization module (Syn-Server) updates mirrors according to the data provided by Syn-Client and the Traffic Monitor module which monitors network traffic between the smartphone and the IP network.

The main critical issue is that mirror servers are not designed for data processing and because of that only limited services (i.e. file caching, file scanning) can be provided compared to the variety of services in cloud data centers.

In the proposed framework, antivirus scanner application can be deployed as a service on the mirror server, and the application can access the file system on mirrors. The benefits of sending antivirus scanner to the mirrors are significant:

1. It saves battery power on smartphones.
2. Scanning will not affect the common usages of smartphone devices since the CPU and I/O intensive workload are moved to the mirror server.
3. Running the scan on mirror will be much faster than that on the phone due to its limited resources.

4.6. Cuckoo

Kemp et al. present in [19] the Cuckoo framework for computation offloading for smartphones. This framework offloads mobile device applications onto a cloud server using a Java stub model. Cuckoo's objectives were to enhance mobile's performance and reduce battery usage. The framework integrates the Eclipse development tool with the open source Android framework.

In the partitioning step, Cuckoo takes advantage of the existing activity model in Android which makes the separation between the intensive and non-intensive components of the application. This activity presents a graphical user interface to the user, and is able to bind to services. The framework can offload intensive components to any resource running a Java Virtual machine (JVM).

As a preparation, the framework requires the developer to write offloadable methods twice - one for local computations and one for remote computations. For this purpose, a programming model is made available to application developers. This programming model is used for dropped connection, supports local and remote execution, and combines all codes in a single package so the user will have a compatible remote implementation.

Cuckoo is a dynamic offloading framework as it takes the offloading decision at runtime and offloads the well-defined components of the application. In case the remote resources are not reachable (i.e. network connection is not available) then the application can be executed on local resources (the mobile device).

4.7. Phone2Cloud

Xia et al. present in [36] a computation offloading framework called Phone2Cloud. The objective was to improve energy efficiency of smartphones and improve the application's performance. Unlike the previous frameworks, authors focus on conducting a fully quantitative analysis on energy saving of the system by conducting application experiments and scenario experiments.

Phone2Cloud is a semi-automatic offloading framework. In order to run applications on the cloud and receive the results, applications need to be manually modified during preparation step to make it possible to be executed on cloud servers.

The offloading decision is based on a static analysis while considering user's delay-tolerance threshold.

For delay tolerant applications, the framework uses a simple model to expect WiFi connectivity.

The threshold is defined based on predictions to delay transfers in order to offload more data on WiFi while respecting the application's tolerance threshold [42]. The framework will wait for WiFi (only if 4G savings are expected within

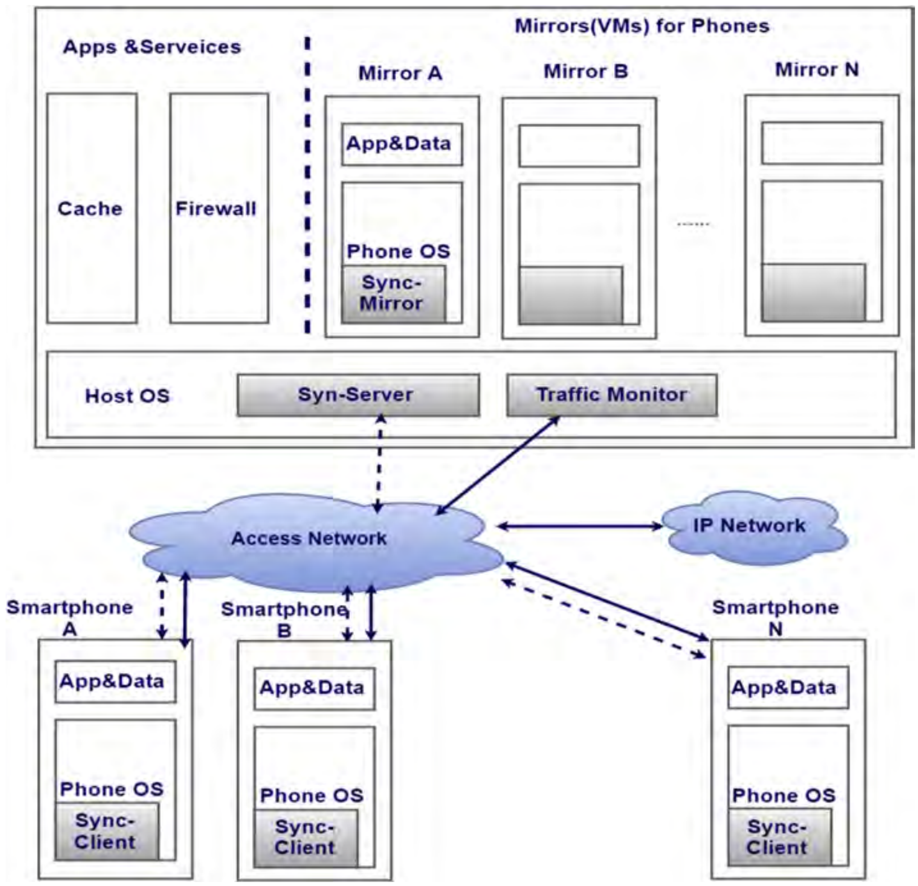

Figure 12 Mirror server architecture (adapted from [38]).

the application's delay tolerance) to become available, rather than sending data immediately.

The framework can offload the whole or part of an application to a cloud server. The prototype of Phone2Cloud is implemented for Android and Hadoop environment (to serve as a cloud). It consists of several components, including an offloading decision engine, a local execution manager, a bandwidth monitor, a resource monitor, an execution time predictor, a remote execution manager, and an offloading proxy that links the offloading decision engine to remote execution manager.

The decision engine is built in order to analyze the power consumption due to offloading. Before execution, two types of comparisons are made:

(1) The average execution time of the application running on the SMD is compared with the user's delay-tolerance threshold.
(2) The power consumption to run the application on the SMD is calculated and compared with power consumption required to run the same application on the cloud.

First, average execution time and user's delay-tolerance threshold are compared. If user's delay-tolerance threshold is smaller than average execution time then the application is offloaded to the cloud. Otherwise, the decision engine checks whether power consumption to run the application on the cloud is greater than power consumption to run the application on the SMD. If it is the case, the application is executed locally. Otherwise, the application is offloaded to the cloud.

An illustration of the architecture of Phone2Cloud is provided in Fig. 13. Phone2Cloud is composed of seven key components.

- Execution time predictor: It is one of the key components in Phone2Cloud. It is used to predict average execution time of an entire application on a mobile device.
- Bandwidth monitor and resource monitor: Bandwidth monitor is used to monitor current bandwidth usage of the network while resource monitor takes care of monitoring CPU workload and other resources. The two monitors serve the offloading decision engine and the execution time predictor separately.
- Offloading proxy: It sends necessary input data to the remote execution manager, receives the results returned by the remote execution manager, and sends back the results to the application.
- Offloading decision engine: It is the core component of Phone2Cloud. Offloading decision engine decides whether to offload the application's components from the mobile device to the cloud server. When it decides to run the application locally, it invokes local execution manager to execute the application. Otherwise, it invokes the offloading proxy to offload computation to the cloud.
- Local execution manager and remote execution manager: The local execution manager is designed to execute the application locally on the SMD. It calls the SMD's operating system, like Android OS, to execute the application. When the remote execution manger receives the required

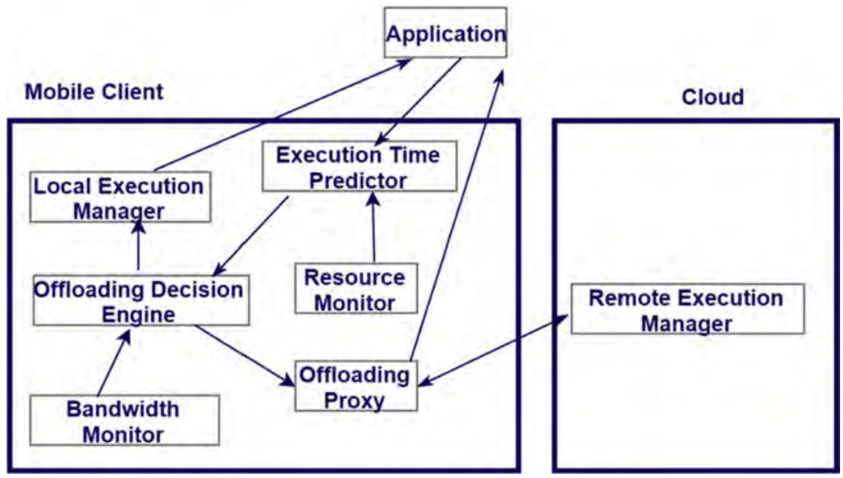

Figure 13 Architecture of Phone2Cloud (adapted from [36]).

input data from the offloading proxy, it runs the offloaded computation on the cloud, and returns results to the offloading proxy.

The authors examine how the energy consumption and execution time of applications will be affected. The evaluated applications are word count, path finder, and sort application. The framework saves energy and improves applications' performance and users' experience of smartphones.

For instance, face finder application costs more energy on smartphone than on a cloud server and the difference between the two costs gets bigger as input grows. The reason behind this is because data transmission costs less energy than running the application locally. Moreover, the energy consumption in smartphones grows faster than that on the cloud server. Thus, face finder application should be offloaded to cloud.

4.7.1. A comparative review of some offloading frameworks

Having reviewed different existing computation offloading frameworks along with their main characteristics, Table 2 presents an overall view about these frameworks and classifies them based on the following attributes:

- Preparation: Any necessary preparations before offloading.
- Partitioning: Partitioning supported or not.
- Decision: Dynamic or static.
- Offloading Mechanism: Mechanism used to offload intensive computations.
- Granularity Level: Granularity Level (i.e. class, method, thread).
- Annotation: Automation of partitioning process (Automatic or manual).
- Contribution: Solved problems?

Annotation is one of the important attributes in partitioning step. It can be seen as a metadata added to the source code. The current partitioning algorithms used in the offloading frameworks can be categorized as (a) automatic and (b) manual.

In automatic annotation, the offloading framework implements automatic annotation by using the profiler to collect the necessary information and annotate the relevant component in the application as an indication of availability of partitioning [6,7].

Manual annotation is performed by the programmers at the design phase. It requires examining the scope of the components of the application at design time. Programmers annotate the components of the application at different granularity such as classes and methods [8,32].

We can notice that some frameworks offload the entire application while other frameworks split the application into its components. Concerning the offloading mechanism, some frameworks encapsulate the offloaded components into a VM or create a VM with exactly the same hardware/software specifications. Other frameworks focus on the code mechanism to offload intensive components. For the annotation attribute, some frameworks use manual annotation while others use automatic one except Phone2Cloud framework which follows a semi-automatic way. Decision offloading is the main attribute of the different offloading frameworks. Some frameworks take the offloading decision at runtime based on a program profiling and program analysis while others take the decision during design or compile time using programmers' annotations and some estimations. A static offloading decision could not adapt to fluctuating network conditions efficiently and depends on programmers' decision. A dynamic offloading decision incurs overhead as it is continuously performed to obtain the latest information.

It can be seen from the presented frameworks that they use different approaches to offload intensive tasks to remote cloud servers. However, none of them use or adopt containers technology such as Linux Containers (LXC). LXC is attracting researchers these days as a lightweight alternative to full machine virtualization such as the common known hypervisors such as KVM or Xen. Recently, research suggests that applications running in containers can achieve approximately same speed in memory, processing and also network throughput as if they were running on a physical machine [41]. LXC is considered as an OS level virtualization where each container has its own environment called a namespace where specific processes are running and isolated from the rest of the system. The usage of containers instead of VM will be a good idea since it is lighter than VM.

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

Table 2 A comparative review of some offloading framework.

Framework	Partitioning	Preparation	Decision	Offloading mechanism	Contribution	Granularity level	Automation	Year
VM Cloudlet [29]	The entire migrating image of the running application is offloaded to the designated remote server while the mobile device provides a user interface and serves as a thin client	It requires the cloning of the mobile device application processing environment to a remote host	Not available	VM: the mobile device transmits all the states of the application to the cloudlet, which applies it to the base VM to launch and execute the VM	Cloudlet-based resource-rich mobile computing	Entire App	Not available	2009
Phone2Cloud [36]	The application can be partitioned or entirely offloaded	Applications need to be manually modified in order to be executed on cloud servers	Static: the offloading decision is based on user's delay-tolerance threshold and static analysis	Code: the remote execution manger gets required input data, it executes offloading computation on the cloud server, and sends back results to the offloading proxy	Enhancement of the application's performance and improvement of energy efficiency of smartphones	Part/Entire App	Semi-automatic	2009
MAUI [8]	Annotate each individual method as local or remote	It creates two versions of a mobile application (for mobile device and cloud). It uses programming reflection to identify which methods are marked offloadable or not	Dynamic: decision is based upon the input of the MAUI profiler and MAUI solver	Code: MAUI does not support executing only portions of a method remotely	Energy-aware code offload	Method	Manual	2010
Mirror Server [38]	The framework does not require a partitioning so the entire application is offloaded	It creates a mirror for a smartphone	Not available	VM: During the copying process, no operation from user is authorized	Reduce the workload and increase the resources of smartphones in a virtual manner	Entire App	Not available	2010
Cuckoo [19]	Partitioning is made based on the existing activity model in Android. The graphical components remain on the mobile device while the services can be offloaded	Destination running a Java VM	Dynamic: method invocations to services are received and Cuckoo framework will then decide whether to offload it or not while checking the availability of the remote resources	Code: the framework receives method calls and evaluates whether to offload the method using heuristics information	Simplifying the development of smartphone applications while benefiting from computation offloading	Method	Manual	2010
CloneCloud [6]	The partitioning is made based on static program analysis and program profiling	A duplicate of mobile device's software stored on the cloud server	Dynamic: threads are migrated from the mobile device to the clone in the cloud	VM: offloaded components of an application are running inside a virtual machine	Elastic execution between mobile devices and clouds while adapting the application partitioning	Thread	Automatic	2011
Jade [28]	An application is partitioned at the class level in Jade. A class must implement one of two interfaces to be offloadable	It checks the application and device status by monitoring the communication costs, work load variation, and energy status	Dynamic: have updated information concerning the status of the application and the device	Code: an offloaded object can be executed on the remote server	An energy-aware computation offloading system	Class	Automatic	2015

5. General issues and challenges in computation offloading for MCC

The selected issues are presented from three perspectives: the resource-intensive structures of the existing frameworks, the security perspective, and the optimal offloading platform.

5.1. Platform diversity

One of the challenges in the current computation offloading frameworks is the diversity and heterogeneity of smartphone architectures and operating systems. This diversity is seen in the following example: MAUI [8] is an offloading framework which is applicable for the .Net framework whereas Mirror Server [38] is a framework which is compatible with the Android platform. A consistent access to cloud services is expected wherein SMDs are enabled to access cloud computing services regardless of the installed operating system or the used hardware. A standardized offloading framework for different smartphone platforms is still a challenging issue in the MCC field.

5.2. Security and privacy in mobile cloud applications

Security of data transmission is an important concern in cloud based application processing. Security and privacy are two crucial concepts that need to be maintained during the offloading process. These concepts can be addressed from different angles: (1) Mobile device, (2) cloud data centers, and (3) during data transmission over the network. Besides all the technologies, there is a great increase in the variety of sophisticated attacks on mobile phones which are the main targets for attackers. Regarding the security in the cloud data centers, threats are basically related to the transmission of data between the different nodes over the network. Thus, high levels of security are expected by both the mobile clients and the cloud providers. In the current frameworks [10,12], binary transfer of the application code at runtime is continually subjected to security threats. Despite the available solutions, strong measures and a secure environment are required for the three entities of MCC model.

In [39], the authors focus on optimizing tasks and computations, and they explore secure offloading of applicable linear programming (LP) computations. In this paper, authors build their work based on the decomposition of the LP computation offloading into public LP solvers running on the cloud and private LP parameters owned by the customer. To achieve an efficient and validate results, the authors focus on the fundamental duality theorem of LP computation and come up with the essential conditions that must satisfied by correct results. Bugiel et al. present in [40] an architecture for secure outsourcing of data and arbitrary computations to an untrusted commodity cloud. The architecture proposed in their approach consists of two clouds (twins): a trusted cloud and a commodity cloud.

The computations are divided in such a way that the trusted cloud is mainly used for critical operations, whereas requests to the offloaded data are processed in parallel by the fast commodity cloud on encrypted data.

However, the idea of dividing operations and handling them by different clouds lead to different difficulties.

For instance, the deployment and maintenance of this architecture of cloud will need clear modifications in the main infrastructure.

The security threat is advancing in a quick manner more than we can keep up with it. Security techniques need to enhance and progress constantly to meet new changes and new offered services. Thus, it is no longer possible to define a security system that would solve all the security threats at once.

5.3. Fault-tolerance and continuous connectivity

In MCC, mobility is one of the most important attributes of SMDs. This is because freedom of movement and autonomy of communication during the consumption of mobile cloud services, are crucial criteria for users' satisfaction. However, there are some constraints that prevent the achievement of seamless connectivity and uninterrupted access to cloud services while on the move. As mobile users move, data exchange rates and network bandwidth may vary. Moreover, users may lose their connection while sending or receiving data; therefore, offloading approaches should be provided with suitable fault-tolerant strategies in order to resend the lost components, minimize the response time, and reduce the energy consumption of mobile devices. It should be noted that the guarantee of a successful execution of offloaded applications is very crucial for mobile users.

5.4. Automatic mechanism

The available computation offloading frameworks still need to be automated. This will help the offloading process to be performed in a seamless fashion while discovering the surrounded environment [5,9,14]. The achievement of such automation is not an easy task as it needs the implementation of a protocol dedicated to finding and discovering services depending on the current context and its constraints.

5.5. Offloading economy/cost

Using cloud infrastructure resources imposes financial charges on the end-users, who are required to pay according to the Service Level Agreement (SLA) agreed on with the cloud vendor serving them. Generally, the operations of content offloading and data transfer between cloud providers incur additional costs on end-users. Therefore, economic factors should be taken into consideration while making the offloading decisions.

5.6. Partition offloading and external data input

At runtime, it is challenging to decide which application components need to be offloaded and to find the suitable server for that. Algorithms answering this problem need resource-intensive effort, which can affect the execution time of the offloaded partitions of the application [13].

Although existing application partitioning algorithms allow an adaptive execution of the application between the mobile devices and the cloud servers, they still do not provide any solution on how to utilize and benefit from the elastic resources in the clouds. This is specifically needed in order to make the

Please cite this article in press as: K. Akherfi et al., Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics (2016), http://dx.doi.org/10.1016/j.aci.2016.11.002

Table 3 Some challenges and open issues in offloading frameworks for MCC.

	Open Issues in offloading frameworks	Challenges to available offloading frameworks
Access a distributed platform transparently	√	√
Continuous connectivity to cloud servers	–	√
Diversity of operating systems in mobile devices along with the variety of their architectures	√	√
Provide an effective execution of a process remotely and returns result to mobile device	√	√

applications scalable when a large number of mobile users need to be served and when the application requires input data that are stored in other remote servers.

Table 3 recapitulates the main challenges to current offloading frameworks and open research issues in MCC. The challenges indicate the issues in the computation offloading frameworks in MCC that still require more elaboration and thorough study, while the open issues specify unresolved problems in current offloading frameworks.

6. Conclusion

This paper discusses three main concepts: (1) cloud computing, (2) mobile cloud computing, and (3) computation offloading. More specifically, it presents existing frameworks for computation offloading along with the various techniques used to enhance the capabilities of smartphone devices based on the available cloud resources. The paper investigates the different issues in current offloading frameworks and highlights challenges that still obstruct these frameworks in MCC. Moreover, the paper shows the different approaches that are used by the frameworks to achieve offloading. Some of these approaches use static offloading while others employ dynamic offloading. Even though there exist a variety of approaches, all of them target the same objective which is the improvement of the smartphone device capabilities by saving energy, reducing response time, or minimizing the execution cost.

We notice that current offloading frameworks are still facing some challenges and difficulties. For instance, lack of standard architectures. This shortage leads to more complications while developing and managing a proposed framework. Finally, it is important to come up with a lightweight paradigm or model that will help to overcome the difficulties and minimizing efforts while developing, deploying, and managing an offloading framework.

We believe that exploring other alternatives, such as introducing a middleware based architecture using an optimizing offloading algorithm, could help better the available frameworks and provide more efficient and more flexible solutions to the MCC users.

Acknowledgments

This work has been funded by the Schlumberger Foundation Faculty for the Future and Technical University of Munich.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[2] R. Barga, D. Gannon, D. Reed, The client and the cloud: democratizing research computing, IEEE Internet Comput. 15 (1) (2011) 72.
[3] B. Butler, Gartner: Cloud Putting Crimp in Traditional Software, Hardware Sales, Networkworld, 2012.
[4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generat. Comput. Syst. 25 (6) (2009) 599–616.
[5] W.-C. Chuang, B. Sang, S. Yoo, R. Gu, M. Kulkarni, C. Killian, Eventwave: programming model and runtime support for tightly-coupled elastic cloud applications, in: Proceedings of the 4th annual Symposium on Cloud Computing, ACM, 2013, p. 21.
[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: elastic execution between mobile device and cloud, in: Proceedings of the Sixth Conference on Computer Systems, ACM, 2011, pp. 301–314.
[7] B.-G. Chun, P. Maniatis, Augmented smartphone applications through clone cloud execution, HotOS 9 (2009) 8–11.
[8] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, ACM, 2010, pp. 49–62.
[9] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, J. Liu, A survey of energy efficient wireless transmission and modeling in mobile cloud computing, Mobile Networks and Applications 18 (1) (2013) 148–155.
[10] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[11] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless Commun. Mobile Comput. 13 (18) (2013) 1587–1611.
[12] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, V.H. Tuulos, Misco: a mapreduce framework for mobile systems, in: Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, ACM, 2010, p. 32.
[13] I. Giurgiu, O. Riva, G. Alonso, Dynamic software deployment from clouds to mobile devices, in: Middleware 2012, ACM, 2012, pp. 394–414.
[14] M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z.M. Mao, X. Chen, Comet: Code offload by migrating execution transparently, in: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), 2012, pp. 93–106.
[15] K. Ha, G. Lewis, S. Simanta, M. Satyanarayanan, Cloud Offload in Hostile Environments, Technical Report, DTIC Document, 2011.
[16] S. Hakak, S.A. Latif, G. Amin, A review on mobile cloud computing and issues in it, Int. J. Comput. Appl. 75 (11) (2013).
[17] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, T. Mudge, A hybrid approach to offloading mobile image classification, in: International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE, IEEE, 2014, pp. 8375–8379.

[18] A. Huth, J. Cebula, The Basics of Cloud Computing, United States Computer, 2011.
[19] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: a computation offloading framework for smartphones, in: Mobile Computing, Applications, and Services, Springer, 2010, pp. 59–79.
[20] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: can offloading computation save energy?, Computer 43 (4) (2010) 51–56
[21] S. Kundu, J. Mukherjee, A.K. Majumdar, B. Majumdar, S.S. Ray, Algorithms and heuristics for efficient medical information display in PDA, Comput. Biol. Med. 37 (9) (2007) 1272–1282.
[22] G.A. Lewis, S. Echeverr'ıa, S. Simanta, B. Bradshaw, J. Root, Cloudlet-based cyber-foraging for mobile systems in resource constrained edge environments, in: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 412–415.
[23] S. Mathew, Overview of Amazon Web Services, Amazon Whitepapers, 2014.
[24] J. Matthews, M. Chang, Z. Feng, R. Srinivas, M. Gerla, Powersense: power aware dengue diagnosis on mobile phones, in: Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare, ACM, 2011, p. 6.
[25] P. Mell, T. Grance, The Nist Definition of Cloud Computing, 2011.
[26] A. Mtibaa, A. Fahim, K.A. Harras, M.H. Ammar, Towards resource sharing in mobile device clouds: power balancing across mobile devices, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 51–56.
[27] Y. Nimmagadda, K. Kumar, Y.-H. Lu, C.G. Lee, Real-time moving object recognition and tracking using computation offloading, in: International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ, IEEE, 2010, pp. 2449–2455.
[28] H. Qian, D. Andresen, Jade: reducing energy consumption of android app, Int. J. Network. Distrib. Comput (IJNDC) 3 (3) (2015) 150–158 (Atlantis Press).
[29] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009) 14–23.
[30] M. Shiraz, E. Ahmed, A. Gani, Q. Han, Investigation on runtime partitioning of elastic mobile applications for mobile cloud computing, J. Supercomput. 67 (1) (2014) 84–103.
[31] M. Shiraz, A. Gani, R.H. Khokhar, R. Buyya, A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing, IEEE Commun. Surv. Tutorials 15 (3) (2013) 1294–1313.
[32] M. Smit, M. Shtern, B. Simmons, M. Litoiu, Partitioning applications for hybrid and federated clouds, in: Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp., 2012, pp. 27–41.
[33] M. Tulloch, Introducing Windows Azure for IT Professionals, Microsoft Press, 2013.
[34] C. Wang, Z. Li, Parametric analysis for adaptive computation offloading, ACM SIGPLAN Notices, vol. 39, ACM, 2004, pp. 119–130.
[35] S. Wang, S. Dey, Rendering adaptation to address communication and computation constraints in cloud mobile gaming, in: Global Telecommunications Conference (GLOBECOM2010), 2010 IEEE, IEEE, 2010, pp. 1–6.
[36] F. Xia, F. Ding, J. Li, X. Kong, L.T. Yang, J. Ma, Phone2cloud: exploiting computation offloading for energy saving on smartphones in mobile cloud computing, Inform. Syst. Front. 16 (1) (2014) 95–111.
[37] K. Yang, S. Ou, H.-H. Chen, On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications, Commun. Mag. IEEE 46 (1) (2008) 56–63.
[38] B. Zhao, Z. Xu, C. Chi, S. Zhu, G. Cao, Mirroring smartphones for good: a feasibility study, in: Mobile and Ubiquitous Systems: Computing, Networking, and Services, Springer, 2010, pp. 26–38.
[39] C. Wang, K. Ren, J. Wang, Secure and practical outsourcing of linear programming in cloud computing, in: INFOCOM, 2011 Proceedings IEEE, IEEE, 2011, pp. 820–828.
[40] S. Bugiel, S. Nurnberger, A. Sadeghi, T. Schneider, Twin clouds: an architecture for secure cloud computing, in: Workshop on Cryptography and Security in Clouds (WCSC 2011), 2011.
[41] D. Bernstein, Containers and cloud: from LXC to Docker to Kubernetes, in: IEEE Cloud Computing 1.3, IEEE, 2014, pp. 81–84.
[42] A. Balasubramanian, R. Mahajan, A. Venkataramani, Augmenting mobile 3G using WiFi, in: Proceedings ACM MobiSys'10, Chicago, 2010, pp. 209–222.