Evidence-aware Mobile Cloud Architectures

Huber Flores, Vassilis Kostakos, Sasu Tarkoma, Pan Hui and Yong Li

Abstract The potential of mobile offloading has contributed towards the flurry of recent research activity known as mobile cloud computing. By instrumenting the mobile applications with offloading mechanisms, a mobile device can save its energy and increase its performance. However, existing offloading mechanisms lack from efficient decision models for augmenting the mobile device with cloud resources on the fly. This problem is caused by the large amount of system's parameters and their scattered values that need to be considered and characterized merely by the device depending on its contextual needs. Thus, the offloading process still suffers from deficiencies that do not allow a device to maximize the advantages of going cloud-aware. In this chapter, we explore the challenges and opportunities of a new kind of mobile architecture, namely *evidence-aware mobile cloud architecture*, which relies on crowdsensing to diagnose the optimal configuration for migrating mobile functionality to cloud. The key insight is that by using the massive parallel infrastructure of the cloud to process big data, it is possible to collect offloading evidence from large amount of devices that is later analyzed in conjunction to infer an efficient configuration to execute a smartphone app for a particular device.

Huber Flores
University of Oulu, Center of Ubiquitious Computing, e-mail: huber.flores@oulu.fi

Vassilis Kostakos
University of Oulu, Center of Ubiquitious Computing, e-mail: vassilis.kostakos@oulu.fi

Sasu Tarkoma
University of Helsinki, Department of Computer Science, e-mail: sasu.tarkoma@helsinki.fi

Pan Hui
The Hong Kong University of Science and Technology. Department of Computer Science and Engineering, e-mail: panhui@cse.ust.hk

Yong Li
Tsinghua University, Department of Electronic Engineering, e-mail: liyong07@tsinghua.edu.cn

1 Introduction

Mobile and cloud computing are two of the biggest forces in computer science [1]. Nowadays, a user relies either on the mobile or the cloud to perform most of the software aid activities, e.g., e-mail, video streaming, image editing, document editing, web browsing, payment, messaging, games, among many others. While the cloud provides to the user the ubiquitous computational and storage platform to process any complex task, the smartphone grants to the user the mobility features to process simple tasks, anytime and anywhere. Therefore, it is logical that the convergence of these two domains into Mobile Cloud Computing (MCC) will lead to the next generation of mobile applications [2, 3].

Generally, mobile devices are able to consume cloud services through specialized Web APIs in a service-oriented manner [4, 5], e.g. REST. A back-end server located in the cloud is a common component of a mobile application, e.g., push notification, Web service, etc. In fact, since the cloud grants dynamic features to the back-end of a mobile architecture, e.g., scalability on the fly, new paradigms such as MBaaS (Mobile Back-end as a Service) are on the rise [1, 6]. Thus, a logical question to answer is *how the cloud can assist the smartphone in creating the post-pc era?*

Since the mobility of the smartphones imposes many limitations in the mobile resources, e.g., processing, storage and energy, among others, several work proposes to offload opportunistically computational tasks from the mobile device to the cloud [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Offloading is a technique that allows a low power device, e.g., smartphones, to outsource the processing of a task, e.g., code, service, job, etc., to a higher capabilities machine [17, 18, 19], e.g., cloud. The potential of the approach for improving the performance and extending the batterylife is widely accepted and proven feasible with latest mobile technologies. However, the technique still suffers from deficiencies caused from the large amount of parameters that need to be configured correctly to optimize the binding between mobile and cloud resources [20], for instance, since last generation smartphones are as powerful as some cloud servers, it is reasonable to bind those devices with even higher capabilities machines, such that the performance is increased instead of decreased. While the offloading gains are improved even further when optimizing the configuration in which a mobile device migrates the tasks to the cloud, it is not a trivial task to find the optimal configuration to offload, mainly because the large amount of possibilities available.

To counter the deficiencies in the offloading process, we explore a new kind of mobile architecture that relies on *crowdsensing* in order to ***diagnose*** the optimal configuration to migrate tasks for a particular device. The key insight is that traces (aka evidence) from the offloading process are collected from the huge amount of devices that outsource tasks to the cloud (community). By using the massive parallel infrastructure to process *big data* [22, 23, 24], the cloud analyzes the evidence to infer the optimal configuration and injects it into each device. Naturally, since the approach relies on data, the improvements are incremental and adaptive based on the amount of data collected. Thus, this type of architecture is defined as an *Evidence-aware Mobile Cloud Architecture* (EMCA).

In this chapter, we start by providing a literature review about how the cloud can assist the smartphone to overcome the limitations imposed by mobility. We then make a comparison of existing solutions and we highlight the differences with our proposed EMCA. Next, we explore the challenges, technical problems and opportunities of an EMCA. Lastly, we discuss about the benefits and drawbacks of the architecture along with our future directions.

2 Mobile Cloud Offloading

Mobile cloud offloading (aka computational offloading, cyber-foraging) has been re-discovered as a technique to empower the computational capabilities of mobile devices with elastic cloud resources. Computational offloading refers to a technique, in which a computational operation is extracted from a local execution workflow, later, that operation is transported to a remote surrogate for being processed externally, and lastly, the result of that processing is synchronized back into the local workflow [17]. Computational offloading has evolved considerably from cloudlets to code offloading.

2.1 Cloudlets

Cloudlets [25] is one of the initial work that propose the augmentation of mobile computational resources with nearby servers in proximity, e.g. hot spots. Cloudlets overcome the problem of connecting to high latency remote servers by bridging the cloud infrastructure closer to the mobile user. The motivation of reducing the latency between mobile device and cloud is to enrich the functionality of the mobile applications without degrading its perception and interaction in environments where network communication changes abruptly. Figure 1 shows a basic cloudlet architecture. The architecture consists of two parts, a client and a server located in proximity, which means that there is no network hopping between the device and the server. A nearby server is managed by a service provider using virtual machines. A virtual machine is migrated from the cloud of the service provider to the nearby server, so that cloud service provisioning (create, launch or delete) for the mobile can occur from the nearby server. Alternatively, the service provider also can migrate a service to other types of infrastructure, e.g. base stations, in order to reduce the communication latency with the device [18].

While a cloudlet overcomes the problems that arise from high communication latency, the deployment of a cloudlet is a complex task, as involves to introduce specialized components or modify existent ones at low level of granularity, e.g., hardware. Thus, its adaptation is neither flexible nor scalable. As a result, many other solutions have been proposed [19]. The goal of these solutions is to optimize the delegation of computational tasks by relying on higher manipulation of the source

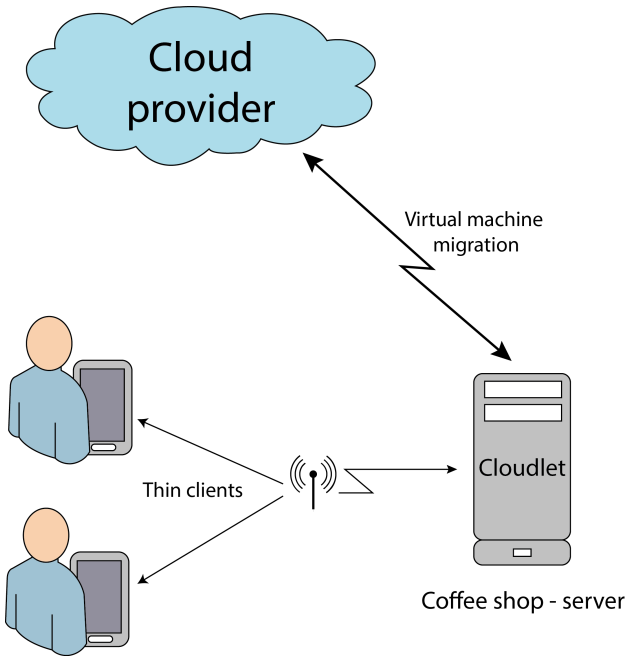

Fig. 1: Components and functionality of a cloudlet system

code of the applications. In this process, computational tasks are delegated to powerful machines at code level as explained in subsection 2.2. Notice that a cloudlet also can be equipped with strategies to offload code [18]. However, the only advantage of using code offloading techniques in a cloudlet model is that the processing of a task can be splitted to multiple devices in a fine-grained fashion, e.g., Method.

2.2 Mobile Code Offloading

Code offloading leverages the small amount of data transferred and the opportunistic high speed connectivity to cloud infrastructure for augmenting the capabilities of the mobile devices [3]. The potential of technique lies in the ability for making the battery life of the smartphones last longer and shortening the response time of mobile applications. Mobile applications are instrumented with code offloading mechanisms for moving a computational task at code level from one place to another. The decision whether to move or not the task from the device for harnessing dedicated external infrastructure is done in the device by analyzing the multiple parameters that can influence the decision to be beneficial or not for the device [10]. The evaluation of the code requires to consider different aspects, for instance, *what code to offload*, e.g., method name; *when to offload*, e.g. RTT (Round Trip Times)

thresholds; *where to offload*, e.g. type of cloud server; *how to offload*, e.g. split code into *n* processes, etc.

Most of the proposals in the field do not cover all these aspects, and thus we describe a basic offloading architecture, which is shown in Figure 2. The architecture consists of two parts: a client and a server. The client is composed of a code profiler, system profilers and a decision engine. The server contains the surrogate platform to invoke and execute code. Each component is described in detail as follows:

1. ***Code Profiler*** is in charge of determining *what to offload*. The profiler characterizes the effort required for the device to execute a portion (*C*) of code —*Method, Thread or Class*. This includes the time of execution and amount of energy required. Based on this characterization, the profiler identifies the code (*OC*) that is candidate to offload. Code can be profiled at different development stages of a mobile application. Thus, we define two types of profilers, manual and automated. Manual profilers are the developers, who select explicitly the portions of code that can be offloaded, e.g., with a code annotation (application is not installed in the device). Automated profilers are runtime processes that analyze the code during runtime using different approaches, e.g., static analysis, history data, etc., and determine the portions of code that are intensive or not for the device (application is already installed in the device).

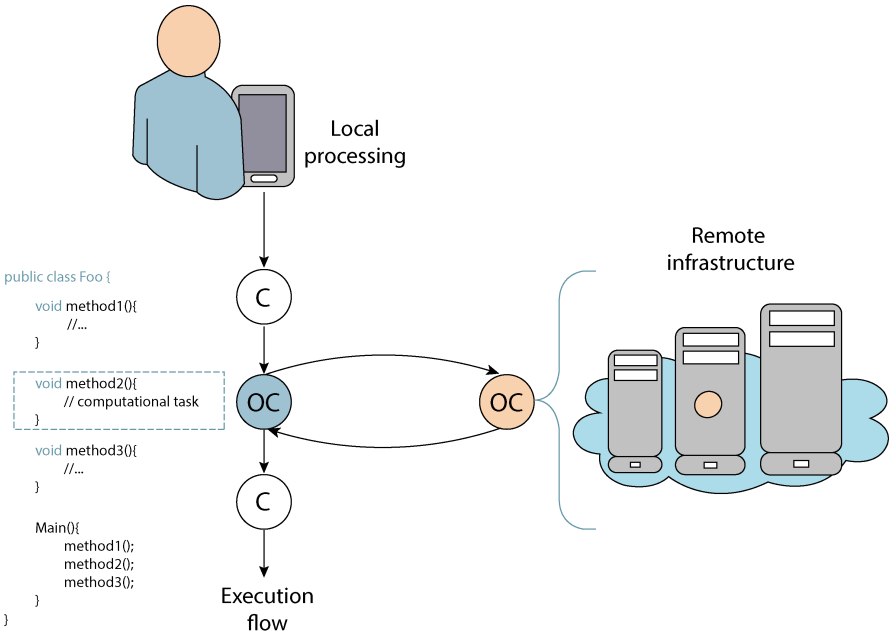

Fig. 2: A code offloading architecture: components and functionalities

2. ***System profilers*** are responsible for monitoring, sampling and characterizing multiple parameters of the smartphone during runtime, such as available bandwidth, data size to transmit, energy required to execute the code, surrogate computational capabilities, etc. These parameters are utilized to quantify whether offloading or not *OC* introduces energy or performance gains for the mobile device.
3. ***Decision engine*** is a reasoner that infers *when to offload* to cloud. The engine retrieves the characterized data obtained by the profilers, and applies certain logic over them, e.g. linear programming, fuzzy logic, markov chains, etc., so that the engine can measure whether the handset obtains or not a concrete benefit from offloading to cloud. The amount of parameters considered in the decision process define the opportunistic context in which a mobile task is offloaded. This suggests that based on the combination of multiple parameters, it is possible to obtain different gains in performance and energy [20]. Proof of this is the way in which existing frameworks characterize with different amount of parameters, the opportunistic moments in which a device offloads to cloud.
4. ***Surrogate platform*** is the computational service located in the proximity of the device or in the cloud, which contains the environment to execute the intermediate code sent by the mobile, e.g. Android-x86, .Net, etc. The computational capabilities of the server are important in an offloading architecture as determine the level in which the task is accelerated [21]. This information is critical to adjust the response time of applications based on the type of device. Ideally, a mobile application must accelerate its execution when offloading rather than slowing down performance.

3 Mobile Offloading Frameworks

In this section, we provide a literature review of frameworks to offload to cloud. Table 1 describes most relevant proposals in code offloading. The table compares the key features of the offloading architectures, namely the main goal, how code is profiled, the adaptation context, the characterization of the offloading process, and how code offloading is exploited from mobile and cloud perspectives. From the table, the main goal defines what is the actual benefit for using the associated framework. The mechanism used to profile code provides information about the flexibility and integrability of the system. The adaptation context specifies the considerations taken by the system to offload. The characterization means whether the offloading system has a priori knowledge or not about the effects of code offloading for the components of the system. Finally, the exploitation highlights the mobile benefits obtained from going cloud-aware, and the features of the cloud that are leveraged to achieve those benefits. Moreover, we can also observe that currently, most of the effort has been focused on providing the device with an offloading logic based on its local context.

MAUI [11] proposes a strategy based on code annotations to determine which methods from a Class must be offloaded. An annotation is a form of metadata that

Table 1: Code offloading approaches from a mobile and cloud perspectives

Code offloading strategies					**Mobile perspective**	**Cloud perspective**
Framework	*Main goal*	*Code profiler*	*Offloading adaptation context*	*Offloading characterization*	*Applications effect*	*Features exploited (Besides server)*
MAUI [11]	Energy-saving	Manual annotations	Mobile *(what, when)*	None	Low resource consumption, Increased performance	None
Odessa [26]	Responsiveness	Automated process	Mobile	None	Applications are up to 3x faster	None
CloneCloud [12]	Transparent code migration	Automated process	Mobile *(what, when)*	None	Accelerate responsiveness	None
ThinkAir [13]	Scalability	Manual annotations	Mobile + Cloud *(what, when, how)*	None	Increased performance	Dynamic allocation and destruction of VMs
COMET [15]	Transparent code migration (DSM)	Automated process	Mobile *(what, how)*	None	Average speed gain 2.88x	None
EMCO [14]	Energy-saving, Scalability (Multi-tenancy)	Automated process	Mobile + Cloud *(what, when, where, how, etc.)*	Based on historical crowdsourcing data	Based on context (Low resource consumption, increased responsiveness, etc.)	Dynamic allocation and destruction of VMs, Big data processing, Characterization-based utility computing
COSMOS [16]	Responsiveness	Manual process	Mobile *what*	None	Increased performance by choosing right surrogate	Resource allocation decided by user
HyMobi [28]	Energy-saving	Manual process	Mobile *what*	None	Energy saving based on social interaction	Resource allocation based on user's social context
Other work [3, 2]	Responsiveness	Manual annotations	Mobile *what, when*	None	Increased performance	None

can be agregated into the source code, e.g. classes, methods, etc. An annotation allows the compiler to apply extra functionality to the code before its called, e.g. override annotations. MAUI uses annotations to identify methods that are resource-intensive for the device. Annotations are introduced within the source code by relying on the expertise of the software developer. Once the code is annotated, MAUI transforms all the annotated methods into an offloadable format. This format equips the methods with RMI capabilities. Since MAUI targets Windows Phones, it is developed using *.NET* framework. Thus, RMI happens by using the WFC (Windows Communication Framework). During application runtime, the MAUI profiler collects contextual information, e.g. energy, RTT, etc., if the MAUI profiler detects a suitable context to offload code, then the execution of the code is delegated to a remote server instead of being performed by the device. While MAUI is successful in saving energy and shortening the response time of the mobile applications, it suffers from many drawbacks. Since MAUI uses code annotations, it is unable to adapt the execution of code in different devices. Thus, the developer is forced to adapt an application to a specific device, which is considered a brute-force approach. Moreover, MAUI suffers from scalability, which means that each mobile that implements MAUI requires to be attached to one specific server acting as a surrogate.

Similarly, CloneCloud [12] encourages a dynamic approach at OS level, where a code profiler extrapolates pieces of bytecode of a given mobile component to a remote server. Unlike MAUI, CloneCloud offloads code at *thread level*. CloneCloud uses static analysis to partition code, which is an improvement over the annotation strategy proposed by MAUI. By using a static analyzer, code can be annotated dynamically. Thus, code to offload is adapted based on the type of device without modifying or changing any implementation of the application. However, code profiling is complicate as its execution is non-determistic. Thus, it is difficult to verify the runtime properties of the code, which can cause unnecessary code offloading or even offloading overhead. Moreover, many other parameters also influence when choosing a portion to code to offload, e.g. the serialization size, latency in the network, etc.

COMET [15] is another framework for code offloading, which follows a similar approach as CloneCloud. COMET strategy puts emphasis on how to offload rather than what and when. COMET's runtime system allows unmodified multi-threaded applications to use multiple machines. The system allows threads to migrate freely between machines depending on the workload. COMET is a realization built on top of the Dalvik Virtual Machine and leverages the underlying memory model of the runtime to implement distributed shared memory (DSM) with as few interactions between machines as possible. COMET makes use of VM-synchronization primitives. Multi-thread offloading accelerates even further the execution of applications in which code can be parallelized.

ThinkAir [13] framework is one which is targeted at increasing the power of smartphones using cloud computing. ThinkAir tries to address MAUI's lack of scalability by creating virtual machines (VMs) of a complete smartphone system on the cloud. Moreover, ThinkAir provides an efficient way to perform on-demand resource allocation, and exploits parallelism by dynamically creating, resuming, and

destroying VMs in the cloud when needed. However, since the development of mobile application uses annotations, the developer must follow a brute-forced approach to adapt his/her application to a specific device. Moreover, resource allocation in the cloud seems to be static from the handset as the device must be aware of the infrastructure with anticipation. Thus, the approach is neither flexible nor fault tolerant. The scalability claimed by ThinkAir is not multi-tenancy, the system creates multiple virtual machines based on Android-x86 within the same server for code parallelization.

Odessa [26] is a framework that focuses on improving the perception of augmented reality applications, in terms of accuracy and responsiveness. The framework relies on automatic parallel partitioning at data-flow level to improve the performance of the applications, so that multiple activities can be executed simultaneously. However, the framework does not consider dynamic allocation nor cloud provisioning on demand, which is a key point in a cloud environment.

History-based approaches are also proposed to determine what code to offload [27]. However, the weak point of history-based approaches is the large amount of time required to collect data, which is needed to produce accurate results. Moreover, these strategies are sensitive to changes, which means that when the device suffers drastic changes, e.g. more applications are installed, the history mechanisms need to gather new data to calibrate again. The size of the data collected in the mobile can also be counterproductive for the device as it steals storage space and processing power [32].

COSMOS [16] is a framework that provides code offloading as a service at method level using Android-x86. The framework introduces an extra layer in a traditional offloading architecture to solve the mismatch between how individual mobile devices demand computing resources and how cloud providers offer them. However, it is not clear how the offloading process is encapsulated as SOA. Moreover, the framework is compared with CloneCloud, which is an unfair comparison as CloneCloud mechanisms offload code at thread level. Other frameworks for computational offloading also are proposed [3], but they do not differ significantly from basic implementation or concept [8, 2, 3]. Other frameworks focus on different issues, such as stability [30] and D2D (Device-to-Device) cooperation [28] among others.

We claim that the instrumentation of apps alone is insufficient to adopt computational offloading in the design of mobile architectures that relies on cloud. Computational offloading on the wild is shown mostly to introduce more computational effort to the mobile rather than reduce processing load [29]. In this context, CDroid [29] is a framework that attempts to improve offloading in real scenarios. However, the framework focuses more on data offloading than computational offloading. As a result, we propose EMCA, which attempts to overcome the issues of computational offloading in practice. EMCA automates the process of infering the right matching between mobile and cloud considering multiple levels of granularity using big data [14, 31].

4 Towards an Evidence-aware Mobile Cloud Architecture

While the need to offload or not for mobile applications is debatable [19], the effectiveness of code offloading implementation in practice shows to be mostly unfavorable for the device outside controlled environments. In fact, the utilization of code offloading in real scenarios shows to be mostly negative [20], which means that the device spends more energy on the offloading process compared to the actual energy that is saved. Consequently, the technique is far away from being adopted in the design of future mobile architectures. In section, we present our EMCA solution. EMCA relies on the smartphones to connect to cloud to characterize all the components of the architecture (Figure 3a), e.g., network communication, cloud-based servers and type of devices, among others at different granularity levels, e.g., code, location, hardware specifications, etc. Once enough data is collected, then it is characterized in the cloud, such that the characterization can be used to create custom configurations for each particular device (Figure 3b).

4.1 Challenges and Technical Problems

Our goal is to highlight the challenges and technical obstacles of developing an EMCA. The issues are described as follows:

- ***Code partitioning approaches*** —Code profiling is one of the most challenging problems in an offloading system, as the code has a non-deterministic behavior during runtime, which means that it is difficult to estimate the running cost of a piece of code considered for offloading. A portion of code becomes intensive based on multiple factors [20], such as user input that triggers the code, type of the device, execution environment, available memory and CPU, etc. Moreover, once code is selected as *OC*, it is also influenced by many other parameters of the system that come from multiple levels of fine-granularity, e.g. communication latency, data size transferred, etc. As a result, code offloading suffers from a sensitive tradeoff that is difficult to evaluate, and thus, code offloading can be productive or counterproductive for the device [33]. Most of the proposals in the field are unable to capture runtime properties of code, which makes them ineffective in real scenarios.
- ***Instrumentation complexity in the mobile applications*** —The adaptation of code offloading mechanisms within the mobile development lifecycle depends on how easily the mechanisms are instrumented within the applications and how effective is the approach in releasing the device from intensive processing. However, implementation complexity does not necessarily correlate with effective runtime usage. In fact, some of the drawbacks that make code offloading to fail are introduced at development stages, for example, in the case of manual code partitioning that relies on the expertise of the software developer, portions of code are annotated statically, which may cause unnecessary code offloading that

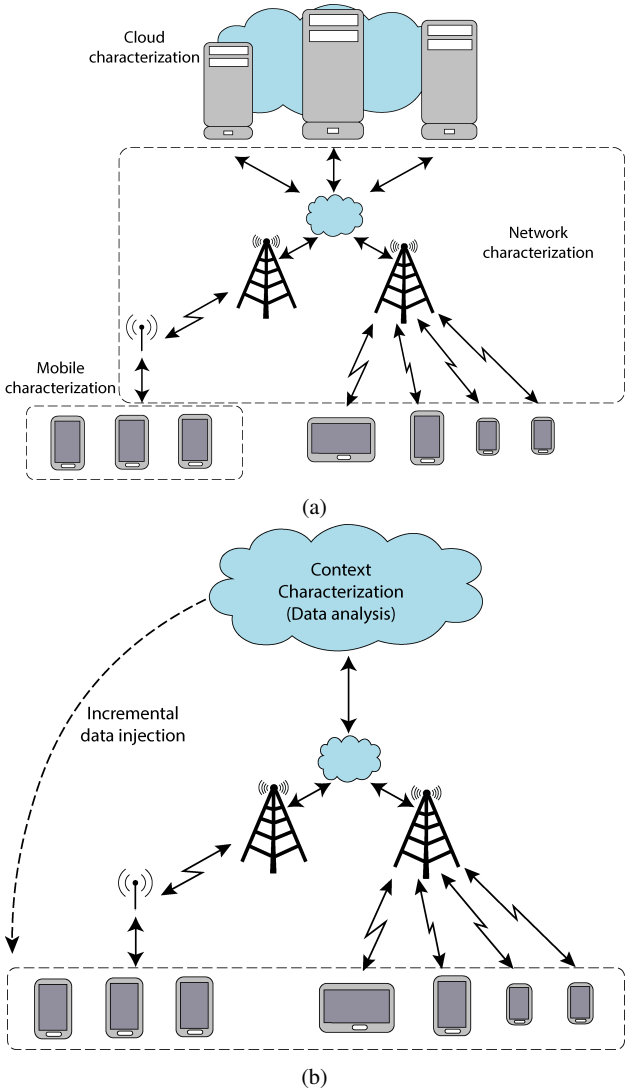

Fig. 3: Evidence-aware Mobile Cloud Architecture. (a) Characterization process of each component. (b) Diagnosis for each device posteriori to the characterization.

drains energy [34]. Moreover, annotations can cause poor flexibility to execute the app in different mobile devices. Similarly, automated strategies are shown to be ineffective and require major low-level modifications in the core system of the mobile platform, which may lead to privacy and security issues.

- ***Dynamic configuration of the system*** —Next generation mobile devices and the vast computational choices in the cloud ecosystem makes the offloading process

a complex task as depicted in Figure 4. Although the savings in energy that can be achieved by releasing the device from intensive processing, a computational offloading request requires to meet the requirements of user's satisfaction and experience, which is measured in terms of responsiveness of the app. Consequently, in the offloading decision, a smartphone has to consider not just potential savings in energy, but also it has to ensure that the acceleration in the response time of the request will not decrease. This is an evident issue as the computational capabilities of the latest smartphones are comparable with some servers running in the cloud, for instance, consider two devices, Samsung Galaxy S (i9000) and Samsung Galaxy S3 (i9300), and two Amazon instances, m1.xlarge and c3.2xlarge. In terms of mobile application performance, offloading intensive code from i9000 to m1.xlarge increases the responsiveness of a mobile application at comparable rates to an i9300. However, offloading from i9300 to m1.xlarge does not provide same benefit. Thus, to increase responsiveness is necessary to offload from i9300 to c3.2xlarge. It is important to note, however, that constantly increasing the capabilities of the back-end do not always speed up the execution of code exponentially, as in some cases, the execution of code depends on how the code is written, for instance, code is parallelizable for execution into multiple CPU cores (parallel offloading) or distribution into large scale GPUs (GPU offloading).

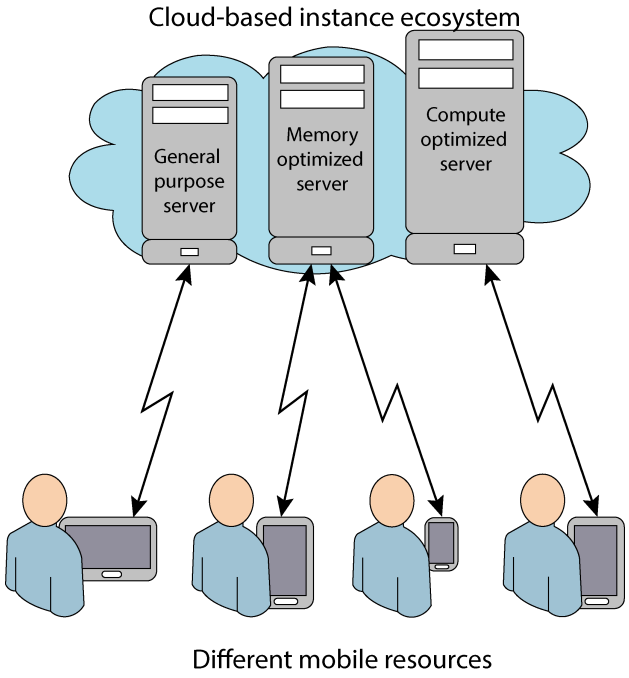

Fig. 4: Characterization of the offloading process that considers the smartphones diversity and the vast cloud ecosystem

- ***Offloading as a service*** —Typically, in a code offloading system, the code of a smartphone app must be located in both, the mobile and server as in a remote invocation, a mobile sends to the server not the intermediate code, but the data to reconstruct that intermediate representation so that it can be executed. As a result, an offloading system requires the surrogate to have similar execution environment as the mobile. To counter this problem, most of the offloading systems proposed to rely on the virtualization of the entire mobile platform in a server, e.g. Android-x86, .Net framework, etc., which tends to constrain the CPU resources and slows down performance. The reason is that a mobile platform is not developed for large-scale service provisioning. As a result, offloading architectures are designed to support one user at the time, in other words, one server for each mobile [13, 16, 39]. This restrains the features of the cloud for multi-tenancy and utility computing. Moreover while a cloud vendor provides the mechanisms to scale Service-Oriented Architectures (SOA) [4, 35, 40] on demand, e.g. Amazon autoscale, it does not provide the means to adapt such strategies to a computational offloading system as the requirements to support code offloading are different. The requirements of a code offloading system are based on the perception that the user has towards the response time of the app [36]. The main insight is that a request should increase or maintain certain quality of responsiveness when the system handles heavy loads of computational requests. Thus, a code offloading request cannot be treated indifferently. The remote invocation of a method has to be monitored under different system's throughput to determine the limits of the system to not exceed the maximum number of invocations that can be handled simultaneously without losing quality of service. Furthermore, from a cloud point of view, allocation of resources cannot occur indiscriminately based on processing capabilities of the server as the use of computational resources are associated with a cost. Consequently, the need of policies for code offloading systems are necessary considering both, the mobile and the cloud.
- ***Utility model for code offloading*** —A code offloaded task is accelerated differently based on the different underlying computational resources that can be acquired in the cloud [21]. While the cost of a server is charged by the cloud vendor based on time usage, e.g., an hour, it is unfeasible to create a bill for computational offloading following the same standard utility model. Since a task can have different levels of resource intensiveness, e.g., Chess, it requires different servers to deal with its specific processing requirements. As a result, a cloud deployment for code offloading comprises not one, but many servers that provisioning their computational resources to a mobile device. In this context, since a particular mobile application can use different cloud servers as surrogates in a single app session, then each offloading task that is offloaded needs to be charged based on the type of server that processed it. Naturally, this implies to change the current utility model of servers to one based on request-type, which introduces an extra level of complexity as it requires the execution of code to be segregated based on runtime properties, e.g., amount of acceleration required.
- ***Evidence usage within the mobile applications*** —Since the characterization process is incremental based on data collected from the community of devices,

evidence about the optimal configuration that is required to execute a mobile application needs to be transferred from the cloud periodically. Consequently, mechanisms to deliver and aggregate evidence need to be developed. Ideally, evidence should be delivered to the mobile device without introducing extra energetic overhead that harms its daily usage. Thus, evidence should be delivered by piggybacking data retrieval from other applications and services.

5 Discussion

In this section, we discuss about the opportunities and drawbacks of exploiting an EMCA.

1. ***Energy-aware offloading as a service for IoT (Internet of Things):*** —
   It is well known that the main goal of MCC is to augment the processing capabilities and energetic resources of low-power devices, e.g., smartphones. To achieve this, applications installed in the devices are instrumented with offloading mechanisms, e.g., code offloading. However, despite of this instrumentation, applications are not aware about the productive or counterproductive effect that can be influenced in the mobile resources by outsourcing a task. For instance, how much the code should be accelerated?, how much energy can be saved? etc.
   In this chapter, we explore how to overcome the problem of determining the context required to offload a task by analysis in the cloud the runtime history of code execution from a community of devices. By relying on the massive computational resources of the cloud to process *big data*, we aim to exploit the knowledge of the crowd. However, many other sources of information collected from a community of devices can provide insight about how to configure the offloading process, e.g., sensor information, user's interaction, etc. To illustrate this, let's consider the following cases:
   Case 1: a smartphone that calculates and transmits its GPS coordinates every time the user uses an application. If the frequency of app usage is high, then the device will run out of energy quickly, e.g., facebook. If we assume that the end service in the cloud stores the data received, the data can be analyzed to build a prediction model in the cloud that suggests when the user changes his/her location. In this manner, the cloud service can be aware about the user's location and can configure the mobile app to recalculate and transmit GPS data when drastic changes of user's location are detected by the model. By implementing this approach, the device can save significant amounts of energy as the computational tasks of calculating and transmitting GPS data are not tied to app usage, but user's movement that is monitored by the cloud.
   Case 2: a low-power device, e.g., Arduino microcontroller, that monitors an environment via sensors, e.g., temperature. Since a client that connects to the microcontroller expects to obtain real time information, the microcontroller senses

the environment regularly. Moreover, in order to provide scalability for multiple users, the environmental information is sent to the cloud, such that any user can access it from there. Naturally, this process requires considerable amount of energy of the device. However, by analyzing the collected data, it is able to equip the cloud service with the awareness to schedule the sensing process of the microcontroller based on opportunistic contexts, for instance, sensing data is likely to be replaced by other sensing data from a nearby device, sensing data can be predicted based on history data stored in the cloud, etc., in any situation, the main goal is to schedule from the cloud, the behaviour of the device, so that the device can be alleviated from unnecessary computational effort. Undoubtedly, it is expected that the change of behavior won't change the quality of service or experience of the user.

2. ***Tuning the fidelity of smartphone apps with mobile crowdsourcing:*** — By characterizing the servers in the cloud, it is possible to identify multiple levels in which offloaded code is accelerated. Thus, we envisioned an approach to accelerate the response time of a mobile application dynamically. The ultimate goal of the approach is to enhance the QoE of the mobile apps in terms of *fidelity*, e.g., face recognition [41]. By improving the QoE, we aim to engage the user in order to increase application usage [43, 37].
   Changing fidelity of mobile apps has been proved to be feasible by collecting data locally in the device [38]. However, this process is slow, because history data is required, and sensitive to changes, because the device is constantly upgrading and installing new apps. Thus, in order to overcome these problems, we envisioned fidelity tuning via data analytics from a community of devices.
   Our idea is that apps are instrumented with mechanisms that capture their local execution at high level, e.g., method name, etc. This data is uploaded to the cloud for analysis. Based on the analysis, the cloud can perform individual diagnosis to each device and suggest optimal fidelity execution of each app installed in the device.
3. ***The effect of computational offloading in large scale provisioning scenarios:*** — While the technique has been proved to be feasible with latest mobile technologies [14], still there are a lot of open issues regarding cloud deployment and provisioning in real scenarios. Previous work have proposed a one server per each smartphone architecture [19], which is unrealistic in practice if we consider the amount of smartphones nowadays and the provisioning cost of constantly running a server for a particular user.
   Besides a few works that focus on scaling up (vertical scaling) a server to parallelize the code of computational requests [26], we have not found architectures that can scale in an horizontal fashion. This clearly can be seen as current frameworks do not take into consideration the utility computing features of the cloud, which is translated into server selection based on provisioning cost.
   We are interested on analysis whether *it is possible to support large scale provisioning for computational offloading?* As a result, we want to study the capacity that cloud servers have to process multiple requests at once while maintaining requirements in code acceleration, which influences directly the response of a

smartphone app. Moreover, we also want to analyze the effect of code acceleration in different cloud servers in order to foster surrogate selection based on utility computing, which can highlight new directions for the design of future mobile architectures supported by cloud computing, e.g., GPU offloading.

4. ***Context-aware hybrid computational offloading:*** — Computational offloading is a promising technique to augment the computational capabilities of mobile devices. By connecting to remote servers, a mobile application can rely on code offloading to release the device from executing portions of code that requires heavy computational processing [42]. Yet, computational offloading is far away to be adopted as a mechanism within the mobile architectures, mainly due to drastic changes in communication latency to remote cloud can cause energy draining rather than energy saving for the device [29, 14]. Moreover, in the presence of high communication latency, the responsiveness of the mobile applications is degraded, which suggests that in order to avoid collateral effects, the benefits of computational offloading can just be exploited in low latency proximity using rich nearby servers [28], which are also known as cloudlets.
   Fortunately, 5G is arising as a promising solution to overcome the problem of high latency communication in cellular networks. 5G fosters the utilization of Device to Device (D2D) communication [30] to release the network from data traffic, and accelerate the transmission of data in end-to-end scenarios. By relying on D2D, and extrapolating features from remote cloud and cloutlets models, we envisioned a context-aware hybrid architecture for computational offloading. Our hybrid architecture introduces the concepts of network and cloud assistance, which can be utilized to coordinate the proximal devices in order to create a D2D infrastructure. Since the computational capabilities of next generation smarphones are comparable with some servers running in the cloud, we believe that multiple mobile devices can be merged together via D2D in order to create dynamic infrastructure in proximity that can be utilized by the devices themselves to share the load of processing heavy computational tasks. Naturally, this introduces new challenges mainly associated to social participation and collaboration.
   Network assistance can be provided by cellular towers (Mobile Edge and Fog Computing [28]). The towers besides routing the communication between end-to-end points can be equipped with the logic to determine which devices are connected geographically close. When devices in proximity are detected, the tower can induce the devices to transmit data via D2D instead of using the cellular tower. The cellular towers can also be utilized to determine closer infrastructure (e.g., base stations), in which the device should be connected to reduce the communication latency, like in the cloudlet model. Similarly, cloud assistance can be utilized to group devices in a D2D cluster. Since devices are offloading to cloud-based servers (e.g., Amazon), the cloud can be equipped with the logic to determine which devices shared a common location. Cloud assistance introduces an extra level of complexity in the system than network assistance, due to a device is forced to send as part of the offloading process, the information about its location (e.g., GPS). However, cloud assistance alleviates

completely the cellular network from computational offloading traffic, as all the process is managed entirely by the cloud.

6 Summary

Mobile and cloud computing convergence is shifting the way in which telecommunication architectures are designed and implemented. Several work have proposed different mobile offloading strategies to empower the smartphone apps with cloud based resources. Yet, the utilization of code offloading is debatable in practice as the approach has been demonstrated to be ineffective in increasing remaining battery life of mobile devices. The effectiveness of an offloading system is determined by its ability to infer opportunistically where the execution of code (local or remote) represents less computational effort to the mobile, such that by deciding *what, when, where and how to offload* correctly, the device obtains a benefit. Code offloading is productive when the device saves energy without degrading the normal response time of the apps, and counterproductive when the device wastes more energy executing a computational task remotely rather than executing it locally. Existing work offer partial solutions that ignore the majority of these considerations in the inference process. Thus, the approach suffers from many deficiencies, which are easily trackable in practice.

By characterizing the offloading process via crowdsensing, we explore the challenges and technical problems to overcome for developing an offloading architecture that learns to diagnose the optimal offloading process of a mobile application.

References

1. H. Flores, “Service-oriented and Evidence-aware Mobile Cloud Computing, *University of Tartu*, Ph.D. thesis, 2015.
2. A. Olteanu, and N. Ţăpuş, “Offloading for Mobile Devices: A survey”, *UPB Scientific Bulletin*, 2014.
3. N. Fernando, S.W. Loke, and W. Rahayu,“Mobile Cloud Computing: A Survey”, *Future generation computer systems*, vol. 29, no. 1, pp.84, 2013.
4. H. Flores and S. N. Srirama, “Mobile Cloud Middleware,” *Journal of Systems and Software*, vol. 92, pp. 82–94, 2014.
5. H. Flores, S.N. Srirama and C. Paniagua, “A Generic Middleware Framework for Handling Process Intensive Hybrid Cloud Services from Mobiles”, in *Proceedings of the ACM International Conference on Advances in Mobile Computing and Multimedia (MoMM 2011)*, (Ho chi minh, Vietnam), Dec 5-7, 2011.
6. M. Mazzucco and M. Dumas, “Achieving Performance and Availability Guarantees with Spot Instances”, in *Proceedings of the IEEE International Conference on High Performance Computing and Communications (HPCC 2011)*, (Banff, Canada), September 2-4, 2011.
7. B. Han, P. Hui, V.A. Kumar, M.V. Marathe, J. Shao, and A. Srinivasan, “Mobile Data Offloading through Opportunistic Communications and Social Participation”, *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, pp. 821, 2012.

8. M. Kaya, et al., "An Adaptive Mobile Cloud Computing Framework using a Call Graph based Model", *Journal of Network and Computer Applications*, vol. 65, pp.12-35, 2016.
9. X. Gu, K. Nahrstedt, A. Messer, I. Greenberg and D. Milojicic, "Adaptive Offloading for Pervasive Computing", *IEEE Pervasive Computing Magazine*, vol. 3, no. 3, pp.66–74, 2004.
10. K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?", *Computer Magazine*, vol. 43, no. 4, pp. 51–56, 2010.
11. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making Smartphones Last Longer with Code Offload," in *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)*, (San Francisco, CA, USA.), June 15-18, 2010.
12. B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud," in *Proceedings of the ACM European Conference on Computer Systems (EuroSys 2011)*, (Salzburg, Austria), April 10-13, 2011.
13. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM 2012)*, (Orlando, Florida, USA.), March 25-30, 2012.
14. H. Flores and S. Srirama, "Adaptive Code Offloading for Mobile Cloud applications: Exploiting Fuzzy Sets and Evidence-based Learning," in *Proceedings of ACM MobiSys Workshop 2013*, (Taipei, Taiwan), June 25-28, 2013.
15. M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: Code Offload by Migrating Execution Transparently," in *Proceedings of USENIX Annual Technical Conference (ATC 2012)* (Boston, MA, USA), June 13-15,
16. C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation Offloading as a Service for Mobile Devices," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2014)* (Philadelphia, PA, USA), August 11-14, 2014.
17. H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile Code Offloading: From Concept to Practice and Beyond", IEEE Communications Magazine, vol. 53, no. 3, pp.80-88, 2015.
18. T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the Cloud to the Mobile User", in *Proceedings ACM MobiSys Workshop 2012*, (Low Wood Bay, Lake District, United Kingdom), June 25-29, 2012.
19. P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the State of Mobile Cloud Computing," in *Proceedings of ACM MobiSys Workshop 2012* (Low Wood Bay, Lake District, United Kingdom), June 25-29, 2012.
20. H. Flores and S. Srirama, "Mobile Code Offloading: Should it be a Local Decision or Global Inference?," in *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2013)*, (Taipei, Taiwan), June 25-28, 2013.
21. H. Flores, X. Su, V. Kostakos, J. Riekki, E. Lagerspetz, S. Tarkoma, P. Hui, Y. Li and J. Manner, "Modeling Mobile Code Acceleration in the Cloud", *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS 2017)*, (Atlanta, GA, USA), June 5-8, 2017.
22. G. Skourletopoulos, C. X. Mavromoustakis, J. M. Batalla, G. Mastorakis, E. Pallis, and G. Kormentzas,"Quantifying and Evaluating the Technical Debt on Mobile Cloud-based Service Level", in *Proceedings of the IEEE International Conference on Communications (ICC 2016)*, (Kuala Lumpur, Malaysia), May 23-27, 2016.
23. G. Skourletopoulos, C. X. Mavromoustakis, G. Mastorakis, J. M. Batalla, C. Dobre, S. Panagiotakis, and E. Pallis, "Big Data and Cloud Computing: A Survey of the State-of-the-Art and Research Challenges", in *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, pp. 23-41, 2017.
24. C. Paniagua, et al., "Mobile Sensor Data Classification for Human Activity Recognition using MapReduce on Cloud", *Procedia Computer Science*, vol. 10, pp.585-592, 2012.
25. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-based Cloudlets in Mobile Computing," *IEEE Pervasive Computing Magazine*, vol. 8, no. 4, pp. 14–23, 2009.

26. M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling Interactive Perception Applications on Mobile Devices", in *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2011)*, (Washington, DC, USA), June 28 - July 1, 2011.
27. A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "SmartDiet: Offloading Popular Apps to Save Energy". *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp.297-298, 2012.
28. H. Flores, R. Sharma, D. Ferreira, V. Kostakos, J. Manner, S. Tarkoma, P. Hui, and Y. Li, "Social-aware Hybrid Mobile Offloading", *Pervasive and Mobile Computing Journal*, vol. 36, pp.25-43, 2017.
29. M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa, "Mobile Offloading in the Wild: Findings and Lessons Learned through a Real-life Experiment with a New Cloud-aware System," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM 2014)*, (Toronto, Canada), April 27 - May 2, 2014.
30. H. Flores, R. Sharma, D. Ferreira, V. Kostakos, J. Manner, S. Tarkoma, P. Hui, and Y. Li, "Social-aware Device-to-Device Communication: A Contribution for Edge and Fog Computing?", in *Proceedings of the ACM International Joint Conference on Pervasive And Ubiquitous Computing (UbiComp 2016): Adjunct*, (Heidelberg, Germany), September 12-16, 2016.
31. A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, "Carat: Collaborative Energy Diagnosis for Mobile Devices," in *Proceedings of the ACM Conference on Embedded Networked Sensor (Systems 2013)*, (Rome, Italy), Nov 11-14, 2013.
32. H. Kchaou, Z. Kechaou, and A. M. Alimi, "Towards an Offloading Framework based on Big Data Analytics in Mobile Cloud Computing Environments", *Procedia Computer Science*, vol. 53, pp.292-297, 2016.
33. G. Chen, B. T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying Energy Trade offs in Offloading Computation/compilation in Java-enabled Mobile Devices", *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp.795-809, 2004.
34. A. Miettinen, and J. K. Nurminen, "Energy Efficiency of Mobile Clients in Cloud Computing", in *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2010)*, (Boston, MA, USA), June 22-25, 2010.
35. H. Flores and S.N. Srirama, "Dynamic Re-configuration of Mobile Cloud Middleware based on Traffic", in *Proceedings of the IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS 2012)*, (Las Vegas, Nevada, USA), October 8-11, 2012.
36. H. Flores, X. Su, V. Kostakos, A. Yi Ding, P. Nurmi, S. Tarkoma, P. Hui, and Y. Li," Large-scale Offloading in the Internet of Things", in *Proceedings of the IEEE Annual International Conference On Pervasive Computing and Communications (PerCom 2017): Adjunct*, (Kona, Big Island, Hawaii, USA), March 13-17, 2017.
37. A. Balachandran, V. Aggarwal, E. Halepovic, J. Pang, S. Seshan, S. Venkataraman, and H. Yan, "Modeling Web Quality-of-experience on Cellular Networks", in *Proceedings of the Annual ACM International Conference on Mobile Computing and Networking (MobiCom 2014)*, (Maui, Hawaii, USA), September 7-11, 2014.
38. M. Satyanarayanan and D. Narayanan, "Multi-fidelity Algorithms for Interactive Mobile Applications", *Wireless Networks Journal*, vol. 7, no. 6, pp.601-607, 2001.
39. M. Kristensen and N.O. Bouvin, "Scheduling and Development Support in the Scavenger Cyber-foraging System", *Pervasive and Mobile Computing Journal*, vol. 6, no. 6, pp.677-692, 2010.
40. P. Nawrocki and W. Reszelewski, "Resource Usage Optimization in Mobile Cloud Computing", *Computer Communications*, 2016.
41. F. A. Silva, et al., "Mobile Cloud Face Recognition based on Smart Cloud Ranking", *Computing*, pp.1-25, 2016.
42. D. Schafer, et al., "Tasklets:" better than best-effort" computing", in *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN 2016)*, (Waikoloa, Hawaii, USA), August 1-4, 2016.

43. Y. Kwon, et al., “Mantis: Efficient Predictions of Execution Time, Energy Usage, Memory Usage and Network Usage on Smart Mobile Devices”, *IEEE Transactions on Mobile Computing*, vol. 14, no. 10, pp.2059-2072, 2015.