Computational Offloading or Data Binding? Bridging the Cloud Infrastructure to the Proximity of the Mobile User

Huber Flores*, Satish Narayana Srirama* and Rajkumar Buyya†
*University of Tartu, Institute of Computer Science,
Mobile Cloud Lab,
J. Liivi 2, Tartu, Estonia
{huber, srirama}@ut.ee
†University of Melbourne, Department of Computing and Information Systems,
Cloud Computing and Distributed Systems Laboratory (CLOUDS),
Vic - 3010, Australia
rbuyya@unimelb.edu.au

***Abstract*—Mobile and cloud computing are converging as the prominent technologies that are leading the change to the post personal computing (PC) era. Computational offloading and data binding are the core techniques that foster to elastically augment the capabilities of low-power devices, such as smartphones. Mobile applications may be bonded to cloud resources by following a task delegation or code offloading criteria. In a delegation model, a handset can utilize the cloud in a service-oriented manner to delegate asynchronously a resource-intensive mobile task by direct invocation of the service. In contrast, in an offloading model, a mobile application is partitioned and analyzed so that the most computational expensive operations at code level can be identified and offloaded to a remote cloud-based surrogate. We compared in this paper, the mobile cloud computing models for offloading and delegation. We utilized our own frameworks for computational offloading and data binding in the analysis. While in principle, offloading and delegation are viable methods to augment the capabilities of the mobile devices with cloud power, they enrich the mobile applications from different perspectives at diverse computational scales.**

***Index Terms*—Mobile Cloud Computing; Access Models; Code Offloading; Task Delegation; Middleware; Mobile Crowdsourcing**

I. Introduction

Mobile and cloud computing are converging as the prominent technologies that are leading the change to the post personal computing (PC) era. Computational offloading and data binding are the core techniques that foster to elastically augment the capabilities of low-power devices [4], [15], [29], [8], such as smartphones. In the case of computational offloading, the process occurs at code level, a mobile application may be partitioned (e.g. methods, classes) and analyzed *a priori* (at development stages) or *a posteriori* (at runtime) so that the most computational expensive operations can be identified and offloaded. A mobile operation may be offloaded or not, depending on the impact of its execution over the mobile resources. Conceptually, offloading is an optional process, which is preferable if a mobile operation requires high amounts of computational processing and at the same time, low amounts of data need to be sent in the communication. Otherwise, offloading is not encouraged [22] as excessive amount of energy and time is consumed in transmission of data to cloud. Moreover, computational offloading has been identified by previous works to be required mainly by applications that implement graphical rendering, image and video processing techniques (e.g. face detection) [7], [5], [31].

In the case of data binding, the process takes different perspectives. Data binding between cloud and low-power devices may occur by following different access schemas [20]. The most common schema is based on service oriented integration via Web API (aka task delegation), where there is a continuous assumption that cloud is always reachable. Task delegation is utilized to delegate a mobile task asynchronously by direct invocation of the service [8]. This task by nature is time-consuming, programmable and parallelizable among multiple servers, and computationally unfeasible for offline devices (e.g. MapReduce jobs) [33]. Moreover, the delegated task aims at enriching the device with a sophisticated feature (PC-like functionality) instead of alleviating its intrinsic components (e.g. battery) from a resource-intensive operation. This is due to the fact that an operation may become process intensive based on the context of the device (e.g. available bandwidth, transfer data size, etc.), but in contrast, this data binding after been established becomes a compulsory operation rather than optional. This means that the delegation of a mobile task can happen at anytime without analyzing the effort required to establish the data binding.

On the other hand, data binding may be also granted with inference logic that enables to decide when to transfer data between local and remote locations based on opportunistic communication algorithms (aka data offloading) [16]. However, this kind of logic is mainly utilized with the purpose of minimizing the data traffic in a mobile network [23], [17]. Thus, decreasing active communication processes within the mobile. Consequently, saving batterylife for the devices.

Data binding techniques are commonly utilized to 1) augment storage space (e.g. DropBox), 2) replicate and centralize data (e.g. Funambol [27]), and 3) integrate services that involve executing remote algorithms over big data repositories (e.g. Activity recognition - queries from mobiles). Moreover, data binding is utilized within the applications based on architectural choices and communication preferences. Thus, unlike computational offloading, data binding counts with bigger spectrum of applicative cases.

While in principle, computational offloading and data binding are viable methods to augment the capabilities of the mobile devices with cloud power, they focus on different issues and implement different techniques to bring the infrastructure closer to the mobile user. Consequently, these approaches enrich the mobile applications from different perspectives at diverse computational scales. In order to investigate the benefits and drawbacks of delegation and offloading, and define the generic access schemas for mobile cloud, we analyzed in this paper multiple case studies that implement the mentioned approaches. We relied on our own frameworks for computational offloading and task delegation for this analysis. Basically, our analysis mainly tries to overcome questions such as what is the complexity of developing a mobile application that implements code offloading/task delegation?, which strategy seems to provide more richer usability and functionality to the mobile user? and which approach tries to exploit the truly concept of cloud computing?. Moreover, we identified major issues and challenges in current models, and discuss possible solutions about how to design the mobile cloud architectures of the future.

The remainder of this paper is structured as follows: Section 2 examines the current state of the mobile cloud solutions and presents the access models for mobile cloud applications. Section 3 presents an evaluation of the methods by presenting some case studies and finally section 4 concludes the paper with future research directions.

II. Access Models for Mobile Cloud Applications

Recently, there has been a growing interest in adapting the cloud computing paradigm for delegating/offloading mobile operations to the cloud. In the case of mobile delegation, current solutions try to overcome the problems of multi-cloud service integration (at SaaS level) in mashups that can be accessible from the handset. More sophisticated solutions try to address the issues of interoperability across multiple clouds, transparent delegation and asynchronous execution of mobile tasks that require resource-intensive processing, and dynamic allocation of cloud infrastructure [8]. In contrast, most of the research works have proposed solutions to bring the cloud to the vicinity of a mobile [9], [7], [5], [14], [21] from an offloading perspective, and thus addressing partially the issues of determining *what*, *when* and *how* to offload from mobile to cloud. The rest of this section examines each model in detail.

A. Offloading Mobile Operations to Cloud

Code offloading has re-gained a lot of interest towards the development of mobile cloud applications as it can be implemented by the smartphones as a mechanism to offload resource intensive operations to cloud-based surrogates [30]. Basically, in mobile code offloading, a mobile application may be partitioned explicitly for remote execution by a software developer [7] (based on his/her expertise) or implicitly by an automated mechanism [5] (based on code profiling techniques), so that at runtime the marked mobile components are identified and analyzed by an offloading decision engine. The engine is in charge of deciding whether a mobile component (aka mobile operation) is offloaded or not. Basically, a decision engine profiles multiple local aspects of the device (e.g. bandwidth connectivity, size of data, etc.) and applies certain logic over them (e.g. linear programming), so that the engine can measure whether the handset obtains or not a concrete benefit (e.g. extended battery life) from the offloading. Mobile application partitioning is preferable by using an automated mechanism at runtime as it provides flexibility to execute the same application on multiple devices with different hardware properties, and thus avoiding a brute-force mobile development approach, which consists of developing the same application for a specific device as needed.

A common code offloading architecture is represented in Figure 1. This kind of architecture fosters a design, in which, the cloud provides the virtual computational resources (aka instances) and the mobile introduces the partition strategies (e.g. static analysis), the decision logic based on local parameters (e.g. network bandwidth), and the basic implementation primitives (e.g. annotations) that enables to synchronize a mobile application stack with a virtual machine running in the cloud (e.g. Android x86). Prominent works in this domain are described in Table I, considering two perspectives, mobile and cloud. Based on this information, we can clearly distinguish the main aspects of an offloading architecture, such as logic of the mechanisms, context considered for adaptation, offloading effects in the mobile applications, cloud features leveraged in the offloading process, etc. From the table, we can also observe that currently, most of the effort has been focused on providing the device with an offloading logic based on its context.

However, given this context, we can argue that much of the advantages of cloud computing are left unexploited and poorly considered. A cloud does not just mean a virtual machine or a pool of servers which are accessible from the Internet. It has its own intrinsic features like elasticity, utility computing, fine-grained billing, parallelization of tasks, dynamic allocation of resources-based capacity, massive data-intensive processing, etc. So an ideal mobile cloud framework should take advantage of several of these features. Cloud computing may introduce many other dynamic variables (e.g. performance metrics) to current code offloading models that could affect the overall offloading decision process. For instance, performance metrics (e.g. CPU load) of the instance/cluster at the cloud may be useful by the mobile in order to determine 1) whether a server

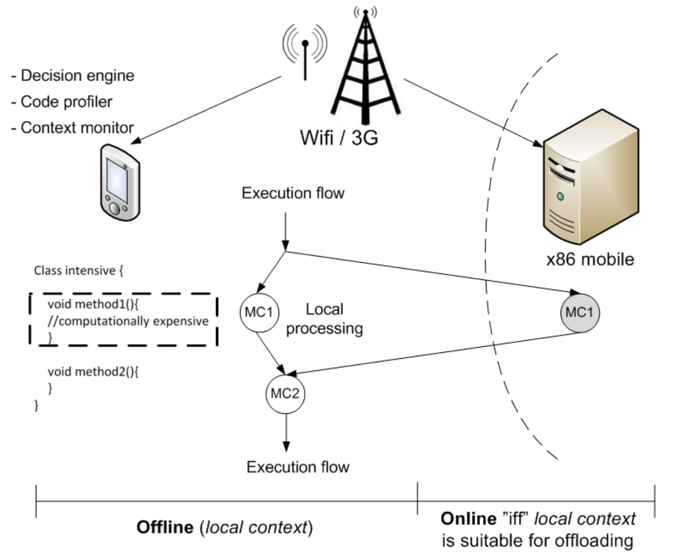

Fig. 1. Offloading model for mobile cloud applications

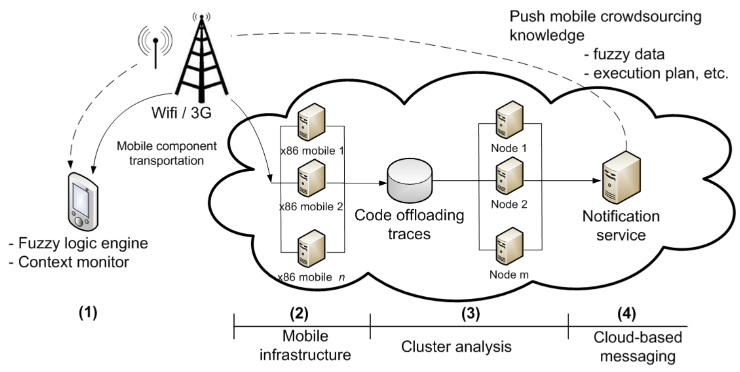

Fig. 2. Evidence-based mobile code offloading architecture

is not that busy so that it can handle an incoming request and 2) a dynamic execution plan that allows to parallelize mobile operations in a single machine with multiple cores or in different machines with a single core. Consequently, a code offloading model should not just target mobility aspects, but also target oscillating changes in cloud infrastructure. Moreover, we think that a mobile cloud architecture must be one that not just increases the storage and computational functionalities of the smartphones when the context of the device is suitable (e.g. low communication latency), but rather, uses its inherent capabilities to process *big data* in order to enhance periodically the mechanisms of the devices with offline cloud analysis based on history mobility data [25], [26], such that each time a device may have opportunity to interact with the cloud again, the handset does it better using refined profiling strategies that allow to increase performance and save energy. Thus, fostering a shared process distributed between mobile and cloud that concentrates in application adaptation based prediction, and in which each related sub-process implements an independent logic.

On the basis of previous assumptions, we have proposed EMCO to counter those issues; its conceptual details are addressed in a different publication [9]. In short, EMCO can be envisioned as a solution to overcome the issues of adaptive application partitioning, offloading decision-making and cloud-aware dynamic resource allocation. Unlike other approaches, EMCO allows to enrich the offloading decision process of a device by exploiting cloud processing capabilities with evidence-based learning methods, over code offloading traces. Basically, EMCO claims that offloading is not a decision process that happens just in the device, but rather offloading is a learning process that involves a global understanding of the complete mobile cloud infrastructure and this understanding may be found in the offloading information traces which are generated by the massive amount of devices that connect to the cloud (aka mobile crowdsourcing). For instance, it could be possible to find which type of instance provides greater offloading benefits to a specific device?, what mobile components to offload based on back-end availability, etc. EMCO fosters a mobile cloud architecture that enhances the decision offloading process of a device by introducing dynamic cloud parameters and exploring code offloading traces. The complete architecture is shown in Figure 2 and consists of 1) a mobile offloading decision mechanism based on fuzzy logic, 2) a virtualized mobile infrastructure (based on Android x86), 3) a repository of code offloading traces along with a cluster to analyze it and 4) a cloud-based messaging framework to push data to the handset asynchronously.

B. Delegating Mobile Tasks to Cloud

Task delegation has become a common operation that is supported by any mobile platform through various mechanisms (e.g. Web sockets, REST-based requests, etc.). Different mobile platforms or versions of the same mobile platform implement different approaches for managing network communication. For instance, Android platform level-10 handles REST-based requests synchronously. In contrast, Android platform higher or equal than level-15 forces the developer to extend any network communication with the *AsyncTask Class* running on a different thread so that it will be executed in the mobile background. However, a cloud request may be time consuming operation (e.g. MapReduce jobs), and thus it requires a proper mechanism to handle the communication, otherwise this can cause an overhead in the mobile resources, in terms of energy (e.g. keeping an open connection while transaction is performed) and operability in the OS (e.g. handset get stuck).

Integration of cloud functionality within the mobile involves adapting different Web APIs from different cloud vendors within a native mobile platform. Vendors generally offer the Web API as an interface that allows programming the dynamic computational infrastructure that support massively parallel computing. Deploying a Web API on a handset is demanding for the mobile operating system due to many reasons like compiler limitations, additional dependencies, code incompatibility etc., and thus in most of the cases the

TABLE I
CODE OFFLOADING APPROACHES FROM A MOBILE AND CLOUD PERSPECTIVES

Code offloading strategies					Mobile perspective	Cloud perspective
Framework	Goal	Code profiler	Offloading adaptation context	Offloading characterization	Applications effect	Features exploited
MAUI	Energy-saving	Manual annotations	Mobile	None	Low resource consumption, Increased performance	None
CloneCloud	Transparent code migration	Automated process	Mobile	None	Performance increased up to 20x	None
ThinkAir	Scalability	Manual annotations	Mobile + Cloud	None	Increased performance	Dynamic allocation and destruction of VMs
COMET	Transparent code migration (DSM)	Automated process	Mobile	None	Average speed gain 2.88x	None
Odessa [28]	Responsiveness	Automated process	Mobile	None	Applications are up to 3x faster	None
EMCO [9]	Adaptation based on context	Automated process	Mobile + Cloud	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

deployment just fails. For instance, Amazon API and typica API [32] allow to manage EC2 instances (run scripts, attached volumes, etc.), jetS3t [19] API provides access to S3/Walrus and GData API [11] enables configuring services such as calendar, analytics, etc. Consequently, software applications that require cloud intercommunication are forced to implement multiple Web APIs. A common task delegation architecture (synchronous communication) is shown in Figure 3. Basically, this kind of architecture aims the integration of cloud functionality which is provided as Web services at SaaS level by using a specialized middleware. The middleware implements the data management logic along with other optimization services (e.g. protocol transformation) that allows the provisioning of cloud functionality to the device. However, there are also other cloud services in each layer of the cloud computing domain (e.g. IaaS) that might enable the mobiles for augmenting their processing capabilities. These services mainly offer parallelization capabilities which are by nature time consuming operations, and thus asynchronous communication is required in the mobile cloud communication.

Asynchronous communication in a mobile cloud environment may be achieved by relying on push technologies (aka notification services) for dealing with remote executions, and thus avoiding the effect of polling caused by protocols such as HTTP (Hypertext Transfer Protocol). However, these mechanisms are considered as black box services which have certain constraints and limitations such as being platform specific (e.g. AC2DM [12]/ GCM [13] for Android, APNS [3] for iOS, MPNS [24] for Windows Phone 7, etc.), the size of the message that can be pushed into the device (e.g. 1024 bytes for Android, 256 bytes for iOS, 4096 bytes for Windows Mobile etc.) and the number of the messages that can be sent to a single handset (e.g. 200,000 for AC2DM). Moreover, such mechanisms are considered to be moderately reliable, and thus are not recommended in scenarios that require high scalability and quality of service. For example: AC2DM simply stops retrying after some delivery attempts.

Asynchronous delegation of mobile tasks to the clouds is useful to overcome the problems of multi-cloud service integration in mashups that can be accessible from the handset, when those include data-intensive operations. A multi-cloud operation consists of delegating mobile tasks to a diversity of cloud services (e.g. from the infrastructure level, platform level, etc.) located on different clouds (e.g. public, private, etc.) and orchestrating (e.g. parallel, sequential, etc.) those transactions for achieving a common purpose. In an asynchronous process, when a mobile application sends a request to access a cloud service, the handset immediately gets a response that the transaction has been delegated to remote execution in the cloud, while the status of the mobile application is sent to local background so that the mobile device can continue with other activities. Once the process is finished at the cloud, a notification about the result of the task is sent back to the mobile, so as to reactivate the application running in the background, and thus the user can continue the activity.

We have studied the delegation of mobile tasks to hybrid clouds in detail and we have developed a Mobile Cloud Middleware [8] framework (MCM) that addresses the issues of interoperability across multiple clouds, transparent delegation and asynchronous execution of mobile tasks that require resource-intensive processing, and dynamic allocation of cloud infrastructure. Moreover, we have developed several successful study cases of data-intensive mobile cloud applications that benefit by going cloud-aware [31]. MCM is introduced as an intermediary between the mobile phones and the clouds for managing asynchronous delegation of mobile tasks to cloud resources (Figure 4). MCM hides the complexity of dealing with multiple cloud providers by abstracting the Web APIs from different clouds in a common operation level so that the service functionality of the middleware can be added based on combining different cloud services. Moreover, MCM enables the development of customized services based on service composition, in order to decrease the number of offloading times needed in a mobile cloud application. Asynchronicity is

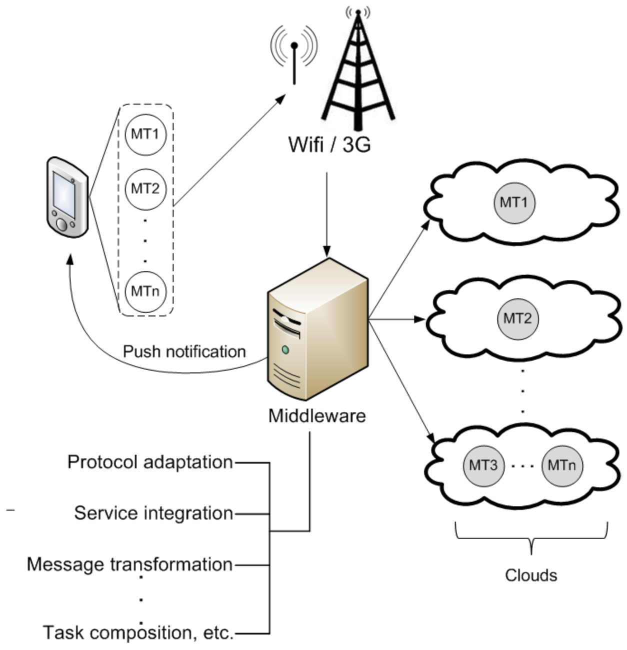

Fig. 3. Delegation model for mobile cloud applications

added to the MCM by using push notification services provided by different mobile application platforms, by extending the capabilities of a XMPP-based IM infrastructure [10] and by using notification services provided by some of the most popular mobile platforms (Android, iOS and Window Phone 7).

MCM is implemented in Java as a portable module based on Servlets 3.0 technology, which can easily be deployed on a Tomcat Server or any other application server such as Jetty or GlassFish. Web APIs are encapsulated using Clojure, and thus are accessed by a common API. This encapsulation guarantees updating deprecated Web APIs with newer versions which are released constantly by the cloud vendor. Moreover, Clojure is also considered due to its distributed nature, which introduces flexibility for scaling the applications horizontally. Thus, augmenting the fault-tolerant properties of the overall system. Hybrid cloud services from Amazon EC2, S3, [1] Google and Eucalyptus based private cloud are considered. Jets3t API enables the access to the storage service of Amazon and Google from MCM. Jets3t is an open source API that handles the maintenance for buckets and objects (creation, deletion, modification). A modified version of the API was implemented for handling the storage service of Eucalyptus, Walrus. Latest version of jets3t also handles synchronization of objects and folders from the cloud. Typica API and the Amazon API are used to manage (turn on/off, attach volumes) the instances from Eucalyptus and EC2 respectively. MCM also has support for SaaS from Facebook, Google and AlchemyAPI.com. We are not analyzing MCM framework in this paper, but rather we are relying on its basic delegation mechanism in order to determine what is the computational gain achieved by the mobile resources through this model.

III. Case Studies

In this section, we compared the models described previously. The comparison is conducted by analyzing two frameworks. MCM is utilized for delegation and an annotation framework based on java similar to [7] for offloading. In the case of delegation, we have developed a simple mobile cloud application in Android platform that delegates a steganography process to the cloud by uploading a picture from the handset via the MCM. A steganography process consists of hiding information of one object within the properties of another object, without affecting the visual representation of this last one for the human eye. The aim of the application is to merge two images into one (encode data) or to extract one image from another (decode data). Notice that more sophisticated applications can be envisioned in multiple scenarios by using MCM delegation [8].

When the mobile application tries to upload the picture for applying the MapReduce steganography process, it sends the request (REST-based) to the MCM along with the picture which is located in a bucket. Once the MCM has received the data, then it sends an acknowledgement back to the mobile, notifying that the process has been started and then releases the mobile so that the device is free to perform other activities. At the MCM, the transactional process consists of creating the internal adapter for starting a new instance (Amazon) using typica API [32], and to attach the picture (located in a bucket) to the instance so that the requested process (encode or decode) can be applied to the image.

Hiding process itself is based on the least-significant bit substitution as shown in Algorithm 1 (Image encoding). One byte from the user's image is saved in three pixels of the base image, 8 bits of one byte are saved in 8 least-significant bits of 8 bytes, leaving the ninth bit to indicate whether there is more hidden data in next bytes or not. Ninth bit information is useful when analyzing the image for extracting information purposes as shown in Algorithm 2 (Image decoding). Change of the least-significant bit in every byte of an image is not trivially recognizable by human eye due to the fact that in RGB color model, there are total of 16,776,216 color variants (2^{24}), but human eye can only distinguish about 10 million different colors. At the end of any process (encode or decode), once the job has completed, result is put to server's static file folder and a link (URL) to the image is provided to mobile user. The URL can be used by the user for downloading and sharing the image with other users.

In the case of offloading, we implemented multiple mobile components (methods) that require data-intensive processing for the device. Thus, we implemented five methods within different Android projects, a *matrix multiplication* method that calculates the product of two 16x16 matrices, a *Fibonacci* method that calculates the Fibonacci number of 200000,a *Quick sort* method that sorts an array of 999999 positions filled with Random numbers, a *Bubble sort* method that orders an array of 9999 positions and a *NQueens* method that implements an heuristic puzzle that calculates how to

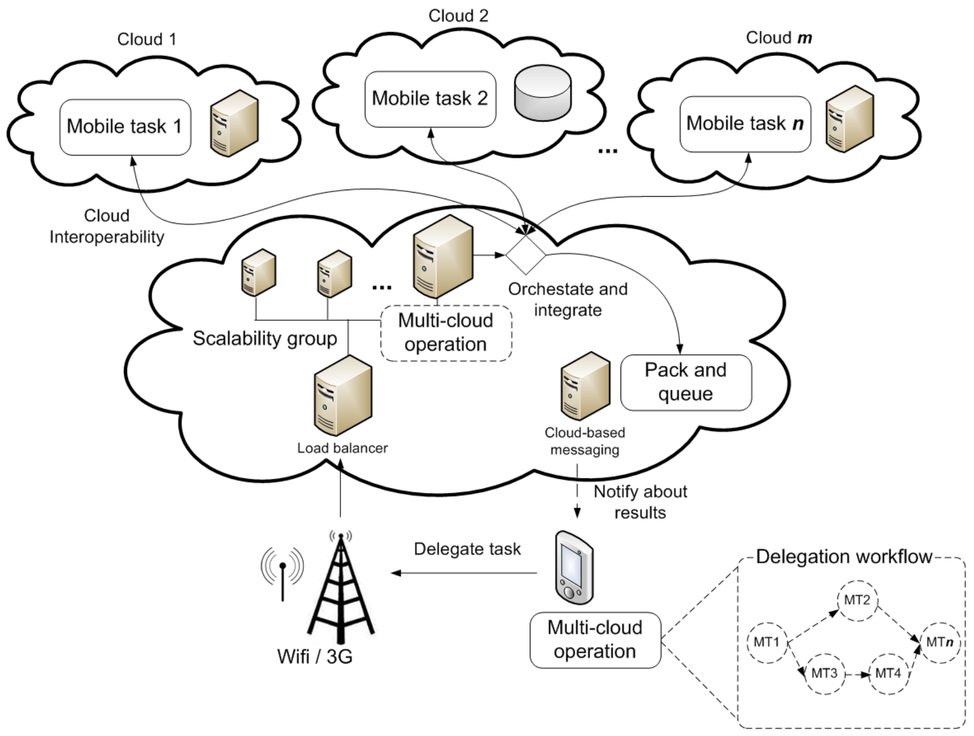

Fig. 4. Asynchronous mobile task delegation for resource-intensive operations

Algorithm 1 Image encoding

```
• Map
  Require: baseImagePath, secretImagePath
    imageBytes = readImage(secretImagePath)
    for each byte in imageBytes do
      secretImageByte = [byteIndex(byte), byteValue(byte)]
      emit(baseImagePath, secretImageByte)
    end for
• Reduce
  Require: baseImagePath, list <secretImageBytes >
    sortedBytes = sortByIndex(secretImageBytes)
    for each secretByte in sortedBytes do
      holderBytes = getThreeBytesFromBaseImage()
      encodeData(holderBytes, secretByte)
    end for
```

Algorithm 2 Image decoding

```
• Map
  identity function
• Reduce
  Require: imageName, imagePath
    image = readImage(imagePath)
    pixels = readPixels(image)
    while true do
      hiddenNineBytes = readThreePixels(image)
      data = readEightLeastSignificantBits(hiddenNineBytes)
      writeToOutput(data)
      if readNinthLeastSignificantBit(hiddenNineBytes) = 1 then
        break
      end if
    end while
```

place *n* queens on an *n* x*n* chessboard. These methods were annotated with the identifier @*Cloud*, later our framework was utilized to read the annotations and create an offloadable version for that particular method. In this process, the class that contains the method is parsed, the annotated method is identified and then a new class with distinct name is created. This new class contains the routines to establish the connection with a server running a Dalvik x86 virtual machine and to push the method for remote execution. At runtime, the mobile application decides based on available bandwidth whether to execute the offloadable or the local definitions.

For both cases, the mobile device considered for running the applications is a *Samsung Galaxy S2 i9100* with Android 4.1, 32 GB of storage, 1 GB of RAM and support for Wi-Fi 802.11 a/b/g/n. Moreover, the device is connected via Wi-Fi to a network with an upload rate of $\approx$ 1409 kbps and download rate of $\approx$ 3692 kbps, respectively. At cloud level, MCM is configured to run on Amazon EC2 infrastructure using a small instance (1.8 GB of memory and up to 10 GB of storage). Dalvik x86 is built from source using Android Open Source Project (AOSP) [2] targeting a x86 architecture. Offlodable components were executed in different instances in order to show how cloud infrastructure may affect the offloading process. Amazon was used as cloud provider and

TABLE II
EXECUTION TIME OF THE MAPREDUCE ALGORITHMS

MapReduce algorithm	Execution time (msec)
Encode	45000
Decode	76000

instances considered were micro (613 MB of memory, EBS storage and 1 virtual core with up to 2 EC2 Units), small, medium (3.75 GB of memory, up to 410 GB of storage and 1 virtual core with 2 EC2 Units) and large (7.5 GB of memory, up to 850 GB of storage and 2 virtual cores with 2 EC2 Units). One EC2 computational instance is equivalent to a CPU capacity of 2.66 GHz Intel®Xeon™processor. Servers were running on 64 bit Linux platform (Ubuntu).

A. Results

On the basis of the functional prototype of the mobile cloud application presented, which is based on steganography, we can derive that it is possible to delegate process intensive hybrid cloud services from the smart phones, via the MCM.

$$T_{mcs_a} \cong T_{tr} + T_m + \Delta T_m + \sum_{i=1}^{n} (T_{te_i} + T_{c_i}) + T_{pn} \quad (1)$$

Where, T_{tr} is the transmission time taken across the radio link for the invocation between the mobile phone and the MCM. The value includes the time taken to transmit the request to the cloud and the time taken to send the response back to the mobile. Apart from these values, several parameters also affect the transmission delays like the Transmission Control Protocol (TCP) [6] packet loss, TCP acknowledgements, TCP congestion control etc. So a true estimate of the transmission delays is not always possible. Alternatively, one can take the values several times and can consider the mean values for the analysis. ΔT_m is the extra latency added to the performance of the MCM. T_{te} is the transmission time across the Internet/Ethernet for the invocation between the middleware and the cloud. T_c is the time taken to process the actual service at the cloud. $\cong$ is considered in the equation as there are also other timestamps involved, like the client processing at the mobile phone. However, these values will be quite small and cannot be calculated exactly. The sigma is considered for the composite service case, which involves several mobile cloud service invocations. However, in other cases the access to multiple cloud services may actually happen in parallel. In such a scenario, the total time taken for handling the cloud services at MCM, T_{Cloud}, will be the maximum of the time taken by any of the cloud services ($Max_{i=1}^{n}(T_{te_i} + T_{c_i})$). Finally, T_{pn}, represents the push notification time, which is the time taken to send the response of the mobile cloud service to the device. With the introduction of support for push notification services at the MCM, the mobile phone just sends the request and gets the acknowledgement back once the multi-cloud operation is performed. However, in this case, the delays completely depend on external sources like the latencies with GCM/APNS/MPNS frameworks and the respective clouds.

Results for delegation are shown in Table II. The value of T_{tr} + ΔT_m is quite short ($<$ 870 msec), which is acceptable from the user perspective. So, the user has the capability to start more data intensive tasks right after the last one or go with other general tasks, while the cloud services are being processed by the MCM. The total time (workflow) taken for handling the cloud services at MCM, T_{Cloud} ($\sum_{i=1}^{n}(T_{te_i} + T_{c_i})$), is also logical and higher as expected. Moreover, it depends directly from the underlying resources of the cloud, which can be configurable dynamically by the middleware. The T_{pn} varies depending on current traffic of the GCM service and has an average of $\approx$4.5 seconds.

In the case of offloading, deriving a model is rather simple as the communication follows a standard client/server design, where latencies to consider are mainly focused on the network bandwidth, which by default, in an offloading process is always good as the mobile local profilers detect when communication is suitable to offload. Moreover, server processing happens in milliseconds (as shown in Table III). Consequently, we do not provide such derivation.

B. Discussion

There are many benefits and drawbacks that emerge from implementing offloading and delegation mechanisms within the development of mobile cloud applications. We highlight key differences in approaches, implementation effort, usability, and richness of the mobile applications.

- Offloading is preferable than delegation as mobile applications can be executed in standalone mode if there is not available connection to the cloud. Thus, making delegation not suitable for contexts out of network communication.
- Delegation enriches the mobile applications with more sophisticated functionality than offloading. Even though, mobile components at Class-method level can be offloaded, the compiler limitations of the mobile virtual machines (VMs) unable to implement complex routines within the mobile. For instance, the Dalvik virtual machine of Android offers just a set of java functionality. Consequently, the richness of the language cannot be exploited and libraries such as jclouds [18] or typica can not be executed on mobile platforms.
- Offloaded mobile components require less execution time than delegated mobile tasks. Consequently, we can argue that delegated mobile tasks do not provide suitable interactivity to the mobile users. However, natively a mobile platform supports and implements for some processes this kind of behavior. Thus, it is not trivial that a mobile application may need long waiting times for completing an operation.
- There are multiple tradeoffs between offloading and resource augmentation. Thus, a mobile application is potentiated by cloud based on its goals (e.g. Energy-saving, responsiveness, etc.). However, we believe that

TABLE III
MOBILE COMPONENTS EXECUTED IN MOBILE AND CLOUD (USING DIFFERENT INSTANCE TYPES)

		Execution time (msec)				
Mobile component	Data size (bytes)	Device (i9100)	Micro	Small	Medium	Large
NQueens	1159	149.30	3.90	3.04	3.41	0.11
MultiplyMatrices	1887	2.69	0.14	0.215	0.204	0.18
Fibonacci	871	81.44	30.70	198.28	31.09	29.34
QuickSort	1485	1653.77	1076.55	2551.83	1254.43	1244.25
BubbleSort	994	3077.74	16850.34	6966.26	2842.39	2428.23

through characterization of an offloading operation, a mobile application can be adapted based on the context, such that a specific tradeoff can be applied at specific context, in order to obtain the maximum benefit each time the device goes cloud-aware.

- Delegation fosters a model, in which, mobile applications are enriched with the variety of cloud services provided on the Web, and thus this allows to create new business opportunities and alliances.
- The effort required to develop a mobile application that follows a delegation model is greater than an application that uses offloading. By default, a mobile architecture for delegation is highly distributed and multi-functional. Thus, it is complex to maintain.
- Different offloading frameworks provide different granularity regarding the definition of mobile components. Currently, mobile components can be offloaded at Class [15], Class-method [7], [21], [9] and Thread level [5], [14], [28]. Each of these levels require a specialized back-end running in the cloud (e.g. Android x86). Moreover, each strategy enriches the mobile application at different performance rates. Refer to Table I for more detailed information.
- Asynchronous delegation suffers from reliability as notification services do not ensure quality of services for delivering messages. However, notification mechanisms are highly integrated with mobile platforms, and thus the mechanisms are optimized to work using low resource consumption.
- Code offloading may fail in some cases, as the current scope utilized by most of the proposed work to characterize an offloading operation is not enough to measure a real benefit for the handset. This can easily be realized as 1) mobile components share a non-deterministic behavior, which makes complex the process of evaluating their impact at runtime (e.g. input variability), and 2) cloud infrastructure play an important role in the overall system. Moreover, next generation technologies for mobiles are computational comparable with some instances running on the cloud. For example, Samsung Galaxy S3 computational power is similar to a micro instance running in Amazon. Consequently, this discourages offloading as a mechanism for increasing the performance of the mobile applications. As explained in previous sections, to counter these issues, we proposed "Evidence-based mobile code offloading" (EMCO) approach [9].

IV. CONCLUSION AND FUTURE DIRECTIONS

In the emerging world of mobile computing, a rich mobile application is one, in which through a soft real-time interactivity, huge amounts of information is processed and presented to the user as a single result. Performing such tasks in a mobile phone is difficult due to the limitations in energy and storage. Thus, computational offloading or task delegation is needed for extending the capabilities of the mobile applications, in order to cover high user demands in functionality. Latest developments in cloud computing offer perfect platform for pushing these process intensive tasks to the cloud.

Computational offloading have been proven feasible with latest mobile technologies (e.g. Android), mostly due to virtualization technologies and their synchronization primitives, enabling transparent migration and execution of intermediate objects. There are multiple tradeoffs between offloading and resource augmentation, and thus offloading adaptation happens by focusing on the goal of the mobile application (e.g. responsiveness). Moreover, computational offloading seems to be necessary just for applications that implement media and image processing functionalities, as those involve processing mobile components that require the execution of many computational steps, like in the case of mobile games. Data binding on the other hand, has a bigger spectrum of applicative cases that can be exploited to augment the capabilities of the mobile applications with sophisticated functionality (PC-like). Moreover, task delegation foster the utilization of cloud in a service-oriented manner, which requires architectures that support high availability, fault tolerance and scalability. Consequently, we believe that mobile cloud architectures of the future have to be designed to fit multiple data binding perspectives.

We compared in this paper, the mobile cloud models for offloading and delegation. We utilized MCM and an offloading framework based on annotations in our analysis. While in principle, offloading and delegation are viable methods to augment the capabilities of the mobile devices with cloud power, they focus on different issues that need to be solved in order to reduce the gap between mobile and cloud. As future directions, we propose investigation on how to support high performance and elastic (scale-out and scale-up) capability for applications via Cloud middleware architectures for code offloading and task delegation in a seamless manner to meet users' quality of service requirements.

Furthermore, from an offloading perspective, we envisioned the exploration of augmented reality mobile applications, which are computationally fed by cloud, such that computational provisioning can improve the perception and responsiveness requirements that emerge from mixing the actual context with the digital device in real-time. Moreover, the exploration of these issues also involve the study about how to deploy virtualized mobile platforms in any cloud architecture in a transparent way. A virtualized mobile platform in the cloud is an architectural requirement for code migration at Class-method level.

Finally, from a data binding perspective, the mobile cloud convergence is also establishing the initial basis (e.g. Mechanisms, access models) that leads to the era of the Internet of Things (IoS). In this new paradigm, firstly, the concept of mobile user is expanded radically to include any entity that is able to transmit and receive data, such that an entity can be address as an object. Secondly, cloud is utilized to spread the pervasive presence of the objects and to support the availability of that presence continuously at high demand scales. Thus, we believe that the study of data binding mechanisms to support continuous sensing will be a key factor that enables to connect any low-power device to cloud.

V. Acknowledgments

This research is supported by the Tiger University Program of the Estonian Information Technology Foundation, the European Regional Development Fund through the EXCS, Estonian Science Foundation grant ETF9287, Target Funding theme SF0180008s12, and the European Social Fund for Doctoral Studies and Internationalisation Programme DoRa.

References

[1] Amazon, Inc, "Amazon - Amazon Web Services," http://aws.amazon.com/.

[2] Android Open Source Project, "Welcome to Android," http://source.android.com/.

[3] Apple, Inc, "APNS," http://developer.apple.com/library/ios/.

[4] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*. ACM, 2002, pp. 87–92.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[6] D. Comer, *Internetworking with TCP/IP, Volume I, Principles, Protocols, and Architecture*. Prentice hall Englewood Cliffs, NJ, 1995, vol. 3.

[7] 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 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[8] H. Flores and S. N. Srirama, "Mobile cloud middleware," *Journal of Systems and Software*, http://dx.doi.org/10.1016/j.jss.2013.09.012.

[9] H. Flores and S. N. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proceedings of the Fourth ACM Workshop on Mobile Cloud Computing and Services (MCS 2013)*, 2013.

[10] H. Flores and S. N. Srirama, "Mobile cloud messaging supported by xmpp primitives," in *Proceedings of the Fourth ACM Workshop on Mobile Cloud Computing and Services (MCS 2013)*. ACM, 2013.

[11] Google Inc. , "Google Data Protocol," http://code.google.com/apis/gdata/.

[12] Google, Inc, "AC2DM," http://code.google.com/android/c2dm/index.html.

[13] Google, Inc., "GCM - Google Cloud Messaging for Android," http://developer.android.com/guide/google/gcm/.

[14] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "Comet: code offload by migrating execution transparently," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*. USENIX, 2012, pp. 93–106.

[15] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," *Pervasive Computing, IEEE*, vol. 3, no. 3, pp. 66–73, 2004.

[16] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, J. Shao, and A. Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 5, pp. 821–834, 2012.

[17] B. Han, P. Hui, and A. Srinivasan, "Mobile data offloading in metropolitan area networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 14, no. 4, pp. 28–30, 2011.

[18] jclouds, "jclouds - multi cloud library ," http://code.google.com/p/jclouds/.

[19] jets3t, "jetS3t - An open source Java toolkit for Amazon S3 and CloudFront," http://jets3t.s3.amazonaws.com/toolkit/guide.html.

[20] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten, "Access schemes for mobile cloud computing," in *Mobile Data Management (MDM), 2010 Eleventh International Conference on*. IEEE, 2010, pp. 387–392.

[21] 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 *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[22] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[23] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: how much can wifi deliver?" in *Proceedings of the 6th International COnference*. ACM, 2010, p. 26.

[24] Microsoft, Inc, "MPNS," http://msdn.microsoft.com/en-us/library/.

[25] D. Narayanan, J. Flinn, and M. Satyanarayanan, "Using history to improve mobile application adaptation," in *Mobile Computing Systems and Applications, 2000 Third IEEE Workshop on*. IEEE, 2000, pp. 31–40.

[26] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5. ACM, 1997, pp. 276–287.

[27] A. Onetti and F. Capobianco, "Open source and business model innovation. the funambol case," in *Int. Conf. On OS Systems Genova, 11th-15th July*, 2005, pp. 224–227.

[28] 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 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.

[29] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[30] M. Shiraz, A. Gani, R. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1294–1313, 2013.

[31] S. N. Srirama, C. Paniagua, and H. Flores, "Social group formation with mobile cloud services," *Service Oriented Computing and Applications*, vol. 6, no. 4, pp. 351–362, 2012.

[32] typica, "typica - A Java client library for a variety of Amazon Web Services," http://code.google.com/p/typica/.

[33] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: a software platform for .net-based cloud computing," *High Speed and Large Scale Scientific Computing*, pp. 267–295, 2009.