Cloud-Assisted Computation Offloading to Support Mobile Services

Khalid Elgazzar, Patrick Martin, and Hossam S. Hassanein, *Senior Member, IEEE*

Abstract—The widespread use and increasing capabilities of mobiles devices are making them a viable platform for offering mobile services. However, the increasing resource demands of mobile services and the inherent constraints of mobile devices limit the quality and type of functionality that can be offered, preventing mobile devices from exploiting their full potential as reliable service providers. Computation offloading offers mobile devices the opportunity to transfer resource-intensive computations to more resourceful computing infrastructures. We present a framework for cloud-assisted mobile service provisioning to assist mobile devices in delivering reliable services. The framework supports dynamic offloading based on the resource status of mobile systems and current network conditions, while satisfying the user-defined energy constraints. It also enables the mobile provider to delegate the cloud infrastructure to forward the service response directly to the user when no further processing is required by the provider. Performance evaluation shows up to 6x latency improvement for computation-intensive services that do not require large data transfer. Experiments show that the operation of the cloud-assisted service provisioning framework does not pose significant overhead on mobile resources, yet it offers robust and efficient computation offloading.

Index Terms—Computation offloading, service provisioning, mobile services, mobile computing

1 Introduction

The role of mobile devices as service providers is strongly supported by the continuous increase in their capabilities and recent availability of high speed wireless network technologies. The range of services that involve mobile devices providing data are on the rise, ranging from entertainment services, such as online social gaming and networking, to crowdsourcing, such as collaborative participatory sensing as well as services that can be offered on the fly, such as video streaming of a current event. However, the rich functionalities that such applications offer increasingly demand resources beyond the capabilities of inherently resource-constrained devices. The lack of resources places limitations on the types of functionality and services that can be offered.

The elastic resource provisioning of cloud computing promises to bridge the gap between the limited resources of mobile devices and the growing resource demands of mobile services through offloading resource-intensive tasks. However, offloading such tasks does not always guarantee performance improvements. For example, offloading may entail large data transfer between the cloud and the mobile device, which compromises the potential performance benefits and incurs higher latency. In some other cases the mobile device may be unable to afford the energy requirements for such data transfers. In fact, the user might prefer to lower the bar of latency constraints to favor energy savings for some specific applications. Thus, the decision on when to offload the execution of web resources to the cloud becomes a critical issue to the overall performance of mobile services.

In mobile environments, cloud-based resource provisioning extends beyond the public cloud. A mobile system may also offload computations to cloudlets [1] and mobile cloud [2], [3]. Additionally, the static resource allocation to computation offloading is inefficient and does not exploit the opportunities that mobility may offer. For instance, a mobile system may need to re-offload computations due to a network failure. Dynamic binding to resource providers allows mobile systems to offload computations to a resource provider and receive results through a better provider in the close vicinity (e.g. a smart vehicle).

This research presents a distributed mobile service provisioning framework that reduces the burden on mobile resources through the offloading of resource-intensive processes to the cloud. An offloading decision model is proposed to determine whether or not remote execution of a resource request brings performance improvements. The decision making involves selecting the best available resource provider according to the resource availability and network conditions using our proposed *Follow-Me-Provider* scheme. This scheme dynamically allocates resource providers to offload tasks to best suit the task requirements and environment context. The decision maker determines the best execution plan that achieves the highest performance gain.

Our main contributions are as follows:

- We present an enabling mechanism for context-aware computation offloading to support resource-constrained mobile service providers.
- We introduce the *Follow-Me-Provider*, a robust scheme that enables dynamic binding between computing tasks and cloud resource providers.

- *The authors are with the School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada.*
  E-mail: {elgazzar, martin, hossam}@cs.queensu.ca.

Manuscript received 10 Feb. 2014; revised 28 June 2014; accepted 29 July 2014. Date of publication 22 Aug. 2014; date of current version 7 Sept. 2016.
Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2350471

The remainder of this paper is organized as follows. Section 2 gives a brief background on computation offloading and outlines related work. Section 3 presents the proposed cloud-assisted mobile service architecture. Implementation details and experimental validations are given in Sections 4 and 5, respectively. Section 6 presents the performance evaluation and offers a comprehensive discussion. Section 7 provides overhead analysis. Section 8 concludes the paper and highlights future directions.

2 Background and Related Work

Over the past few years, significant study has been done on the resource constraints of mobile devices as computing platforms. In this section, we provide a brief background and review the related work from two perspectives: mobile devices as service providers and computation offloading to augment the capability of mobile devices.

2.1 Mobile Devices as Service Providers

Using mobile devices as service providers is a new trend [4], [5]. Little of the previous research have been dedicated to investigate computation offloading in service provisioning. Weerasinghe et al. [6] studied reliable mobile service provisioning with respect to availability and scalability. The authors propose a proxy-based middleware to bootstrap the performance of mobile services. The proxy acts as a fixed representative to mobile services. This middleware supports service migration where mobile providers may choose to switch to an alternate server due to close proximity or better connectivity. Hassan et al. [7] present a distributed mobile service provisioning framework that partitions the execution of resource-intensive services between the mobile provider and a backend server. The framework offers a distributed execution engine where tasks that require real time access to local resources are executed on the mobile devices, while the remaining processing is offloaded to a remote server. Their partitioning technique relies solely on the available resources of the mobile device. The execution is entirely performed on the mobile device if available resources satisfy the service execution requirements. In contrast, our framework selects the best execution plan with the minimum response time, while satisfying the resource constraints with respect to both execution requirements and user preferences.

2.2 Computation Offloading

Computation offloading transfers processing outside of the mobile device. The objective is to improve the performance, enable advanced functionality, and preserve scarce resources. Offloading may be performed at different granularities ranging from methods and individual tasks [8], [9], [10] to applications [11] and virtual machines [12]. Offloading techniques are categorized into three main categories:

1) *Remote invocation.* This technique requires services to be deployed on the remote host. The offloading device invokes the target service on the remote server using well-known mechanisms such as remote procedure call (RPC) or remote method invocation (RMI). Although this approach is well-supported by APIs, service pre-deployment on remote hosts poses a critical restriction to the ad-hoc nature of mobile cloud systems. For example, mobile terminals may discover resource providers in close proximity that offer IaaS. Such providers could be transient (e.g. a passing smart vehicle), or stationary, such as cloudlets [1]. In such cases, the pre-deployment requirement limits the choices of mobile systems on where to offload computations.
2) *VM migration.* VM migration refers to transferring the entire memory image of a running VM from the mobile system to a cloud infrastructure. VM migration has advantages over other methods as VMs are created through virtualization techniques, which offer isolation and protection for data and processing against malicious entities on the remote host.
3) *Code migration.* Mobile code migration transfers code and data required to carry out a certain task on a remote host. In most cases, the code is small in contrast to the data. However, compatibility issues may arise when migrated code is ported to run on a hosting platform. This method enables computation offloading on a fine granularity level, such as individual functions and methods. VM and code migration are favored in recent works [13].

Several aspects of computation offloading have been studied including the feasibility of offloading, making offloading decisions, and developing offloading infrastructures [14], [15]. For example, *Hyrax* [2] is a platform that supports distributed Android applications based on a version of Hadoop ported to the Android platform. It enables Android applications to access data and offload computing tasks to a mobile cloud formed by heterogenous mobile devices. Huerta-Canepa and Lee [16] propose a virtual mobile cloud computing platform using mobile devices. The framework capitalizes on the availability of mobile devices that are willing to share their computational resources in the close vicinity. Giurgiu et al. [17] present a middleware that can distribute mobile applications between the mobile device and a remote server, aiming at improving the overall latency and reducing the amount of data transfer. The middleware generates a resource consumption graph and splits the application's modules to optimize a variety of objective functions. Cuckoo [18] is a framework that provides a runtime environment for mobile applications to supports dynamic offloading decisions. However, mobile applications need to be re-written according to the Android's *'activity/service'* model.

Clonecloud [10] and *MAUI* [9] are two recent studies on computation offloading that focus on performance gain and energy saving, respectively. *CloneCloud* offers a runtime partitioning approach for mobile applications based on a combination of static analysis and dynamic profiling techniques. *CloneCloud* works at the application-level VM and supports up to the thread granularity. The objective is to speed up the application execution and to reduce energy consumption at the mobile side. In *CloneCloud*, a device clone operates continuously on the cloud and communicates with the mobile device. *MAUI* enables energy-aware offloading of mobile code to a resource-rich computing infrastructure. It aims to alleviate the burden on the limited energy resources of mobile devices while fulfilling the

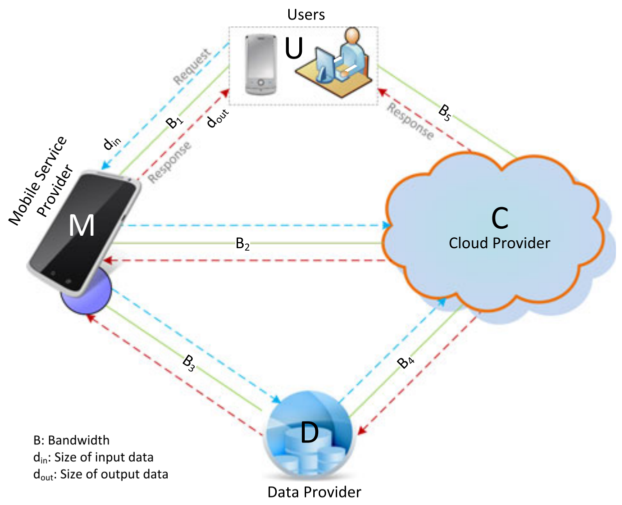

Fig. 1. An abstract view of cloud-assisted mobile service architecture, showing possible interacting entities and context information, where B is the link bandwidth and d_{in} and d_{out} represent the amount of data exchange over a link in both directions.

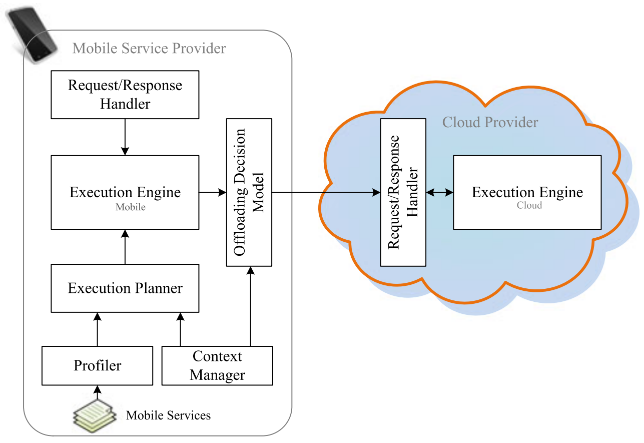

Fig. 2. The architecture of the cloud-assisted mobile service provisioning.

increasing energy demands of mobile applications and services. *MAUI* provides a disconnectivity mechanism that enables interrupted processes to resume execution on the mobile device. However, *MAUI* requires source code annotation by developers to mark which code can be executed remotely and which cannot. It uses these annotations to decide at runtime on the proper partitioning scheme.

In general, the offloading decision is made based on an objective to either reduce the response time, save energy, or strike a balance between both. Offloading techniques make decisions statically based on prespecified rules, or dynamically based on context changes [19]. Dynamic decisions can be made using machine learning techniques [20] or based on analytical models [18], [19], [21], [22]. *Spectra* employs computation offloading to balance between performance and energy consumption. The decision is made based on the resource usage profile of mobile applications and resource availability in the surrounding environment. *Spectra* requires continuous monitoring of local and remote resources and network conditions. Spectra also modifies the application's source code to determine possible partitioning options. *Scavenger* [22] uses various attributes to decide whether offloading of a computing task brings performance gains. These attributes are: powerfulness and utilization of available surrogates, network bandwidth, task complexity, and required data transfer. Kumar and Lu [23] provide an analytical model to determine whether computation offloading can save energy, taking into consideration the resource requirement of the computing task and current network conditions.

To the best of our knowledge, all previous research efforts on mobile computation offloading focus only on the interactions between the mobile system and the support computing infrastructure, within the context of environment conditions, while overlooking the possibility that a third-party data provider may exist. Many current data providers offer their services either through the public cloud itself (e.g. Amazon S3) or connect to the cloud via a high speed interconnect (e.g. Dropbox). Therefore, the data transfer is much faster between the data provider and the cloud, than between the data provider and a mobile device. Hence, adding a data provider to the equation significantly impacts the offloading decision. The framework proposed in this paper addresses this issue when making offloading decisions. Furthermore, it also proposes an agile cloud provider selection algorithm that leverages the mobility of both users and mobile resource providers.

3 Cloud-Assisted Mobile Service Architecture

This paper presents a framework for cloud-assisted service provisioning from resource-constrained devices. The proposed cloud-assisted mobile service architecture involves four key entities: a user, a mobile device, a cloud, and a data provider, as shown in Fig. 1. The user represents the service consumer. The mobile device represents a mobile service provider and acts as the integration point where service execution plans are generated and decisions regarding offloading are made. The cloud is the supporting computing infrastructure that the mobile provider uses to offload resource-intensive tasks. Service operations may involve third-party data processing during the execution of the service functionality, such as weather information or navigation databases. In such cases, data could be fetched from a data provider.

The user sends the service request to the mobile provider. The mobile provider decides on the best execution plan and whether offloading is beneficial. The cloud offers elastic resource provisioning on-demand to mobile providers. The mobile provider may collect the execution results from the cloud and generate a proper response for the use or delegate the cloud to forward the response directly to the user, given that no further processing is required at the mobile side.

The proposed framework encompasses the following major components: *Request/Response Handler*, *Context Manager*, *Profiler*, *Execution Planner*, *Service Execution Engine*, and *Offloading Decision Maker*. Fig. 2 depicts an abstract view of the framework architecture. The functionality of each component is discussed in the following sections.

3.1 Request/Response Handler

Internet users can request access to web services or web contents. A service request could be SOAP/XML for

SOAP-based web services [24] or an HTTP request for REST-based web services [24]. RESTful requests point directly to specific operations that carry out the required functionality whereas SOAP requests associate parameters by which the service execution engine internally maps the request to the appropriate method. The *Request/Response Handler* plays the role of a multiplexer, distinguishing between SOAP requests and RESTful requests as well as differentiating between service and content access requests. The handler forwards the latter directly to the web server whereas the former is sent to the service *Execution Engine* for processing. SOAP/XML service requests are handled by *SOAP Manager* before they are sent to the web server, while HTTP requests for services are analyzed directly by a servlet.

3.2 Profiler

This component is responsible for analyzing the characteristics of various service operations, deployed on the mobile device, in the form of a resource consumption profile that includes the required CPU cycles, memory size, data exchange, potential data transfer, and interactions with local resources. A service may include multiple operations. Each operation can be invoked separately, possibly many times, and perform its functionality independently. The profiler treats each operation as a standalone function. The profiler runs service operations offline to measure the required resources in terms of CPU cycles, memory, data transfer, and access to local physical resources. We instrument these operations to identify the dependencies and interrelationships between them. The profiler then generates a resource consumption profile for each service with a separate section for each operation [19]. The planner module uses the information in the consumption profile to generate possible execution plans.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:env="http://www.example.sample/profile#">

<rdf:Description
 rdf:about="http://service-root/service-name?wsdl">
   <env:serviceName>name</env:serviceName>
</rdf:Description>

<rdf:operation
 rdf:about="http://www.example.sample/profile">
   <env:uri>http://service-roor/Blur</env:uri>
   <env:cpu>16</env:cpu>
   <env:memory>16.2</env:memory>
   <env:energy>1.92</env:energy>
   <env:dependancy>None</env:dependancy>
</rdf:resource>

<rdf:operation
 rdf:about="http://www.example.sample/profile">
   <env:uri>http://service-roor/Blend</env:uri>
   <env:cpu>106</env:cpu>
   <env:memory>19.1</env:memory>
   <env:energy>2.63</env:energy>
   <env:dependancy>None</env:dependancy>
</rdf:resource>
 .
 .
 .
</rdf:RDF>
```

Listing 1. A snippet of a resource consumption profile.

3.3 Context Manager

Mobile devices capitalize on their sensing capabilities to collect real-time context information to make better decisions. The context manager gathers a variety of context information to be used for personalization purposes and adaptive actions [25]. The context manager gathers two categories of context attributes:

- *User context*. This category contains context information related to the user and current runtime environments. This context presents a comprehensive view of the user's current status. User context includes the following attributes:

  Location. The user location determines which remote hosts can be used for offloading. The automatic collection of location information can be obtained in a variety of ways outdoors (e.g. GPS and mobile networks) [26], [27], or indoors (e.g. Received Signal Strength techniques) [28], [29]. Mobility of both devices and resource providers plays a significant role in deciding the offloading strategy. Mobile platforms provide APIs to access the location information. Location-based mobile applications and services use these APIs to obtain the location information on-demand and utilize it internally as required.

  Device profile. This contains the hardware and software specifications of the user's device. The device profile includes the display size and specifications, sensing capabilities, network interfaces, OS platform, etc. These context attributes assist in determining the compatibility of offloading tasks with the remote server specifications. Some of the hardware features also may impact the offloading decision, such as available network interfaces.

  Local resources. Upon receiving a request, the context manager captures on-demand the status of local resources such as: CPU utilization, available memory, battery level, and storage capacity. These attributes are used to evaluate different execution plans to determine the potential performance gain. The context manager also maintains a record of running applications and activities as a load indicator.

  User preferences. Each user has different preferences that might influence the offloading decision. These preferences may change according to the situation. For instance, a user running a navigation application may favor execution plans with more energy saving if battery power is under certain threshold. The same user may be more concerned about latency if they need to find a gas station before the next exit on a highway. The framework maintains user preferences in an XML file with pre-defined tags. The user can update these preferences in an ongoing manner.

- *Environment context*. Environment context encompasses information that characterizes the surroundings of the user. The framework utilizes this context to better make offloading decisions. Context attributes of this category are:

  Remote hosting environments. The mobile system discovers and keeps track of all potential resource

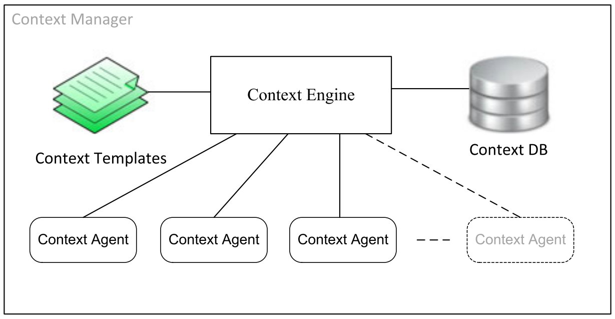

Fig. 3. The schematic architecture of context manager.

providers that can process offloading requests including public cloud, cloudlets [30], mobile cloud farm [2], [16]. Information about available public cloud and cloudlets is provided and updated by the service provider. Upon receiving a service request, the framework sends standard resource discovery requests on-demand over short range communications to discover mobile resource providers in the close vicinity.

Network status. Mobile environments are characterized by intermittent connectivity and varying connection quality. The context manager proactively monitors available network connections, such as wired, WiFi, 4G, Bluetooth, ZigBee [31], and their status. The quality of the connection determines how fast data can be transferred. The network bandwidth and required data play a major role in determining whether offloading is favored over local execution.

Fig. 3 shows the architecture of our context manager. It contains four main components: context engine, context templates, context database, and context agent.

Context engine. The context engine coordinates between different components. It creates and maintains context agents based on registered context templates, gathers context information from agents and stores it in the context database, processes context information if required, and disseminates context data to requesting entities.

Context templates. Templates are configuration files that describe context attributes and any associated constraints and rules. The context engine utilizes these configuration templates to deploy software agents that collect context attributes.

Context database. The context database is a lightweight relational database that maintains various context information. Most mobile-based platforms contain embedded lightweight database engines that support standard relational database functionality. For example, the Android platform uses the SQLite [32] database management system. SQLite requires limited memory at runtime, which makes it a good candidate for mobile systems.

Context agent. The context agent is a software entity that is deployed to gather a specific context attribute. Agents are APIs or web services that could be deployed locally on the mobile system to gather local context information or remotely to report remote context attributes.

Context information and service consumption profiles are used by the offloading decision model to select the optimal execution plan that, in addition to achieving performance gain, satisfies the device constraints and user preferences.

3.4 Execution Planner

The execution planner determines the various possible execution plans for each service operation based on available information about each operation from the service description file and the behavior profile generated by the profiler. Each operation can be executed in a variety of different ways. Possible execution plans are generated based on the sources of involved data objects, interactions between such data and other local resources, and the execution environment. Options include local execution, remote execution or combinations of both. Service developers may annotate particular operations to be strictly executed on mobile systems due to security reasons or privacy concerns such as the case when a provider wishes to ensure full privacy of its users' information. Although current service description standards do not support such annotation, a recent initiative has been proposed by Hassan et al. [7] on how to specify the execution environment in the service description.

Since the functionalities of mobile services are known in advance, execution plans could be generated during the service deployment outside of the mobile service provider (i.e. services are deployed along with their possible execution plans). These execution plans could be generated by using the cloud resources supporting the mobile service. However, our platform supports offloading to mobile cloud and cloudlets on-demand, which means that the service does not know which cloud resources are going to be used during execution. Additionally, service requests might be associated with user security/privacy constraints that govern how the requested functionality is executed. For example, the user requires that his/her credentials are not to be shared with third-parties (e.g., cloud providers). This requires revisiting the execution plans at runtime to exclude plans that violate the request constraints. Therefore, to maintain the generality of our framework, we implement the execution planner component on the mobile device to facilitate the interactions with other components, such as the context manager, when required.

The planner starts with the possibility of performing the execution locally on the mobile system, where the device acquires all the required data for processing and sends back a response to the user. If there is local resource access, the planner generates a plan consisting entirely of remote processing. When offloading is applicable, the planner considers *response forwarding*, where the mobile service provider may delegate the cloud/remote execution environment to forward the response directly to the user. If the requested operation encompasses independent functions that can be carried out separately, further plans of partitioning are considered. Several partitioning strategies are discussed in [9], [10], [17].

The framework enables and disables *response forwarding* through setting the *forward_response* attribute to *'ON'* and *'OFF'*, respectively. Response forwarding is a two-fold benefit: 1) mobile providers do not have to retrieve the response back from the cloud and send it to the user. This decreases the communication overhead and battery consumption on the mobile system and reduces the overall response time to the user, 2) the response will reach the user even if the mobile service provider is down/disconnected for any reason. Response forwarding is performed at the application layer through asynchronous communication protocols. Applications with strict security constraints may disable the *'response forwarding'* service, or offload only to trusted providers.

A plan evaluation is performed at runtime once an invocation request is received at the mobile provider's side. Since the churn of services is low, these evaluations are stored for a short time t_p in case the mobile provider receives multiple requests for the same operations within the time interval t_p. It's uncommon that network conditions (bandwidth B in particular) fluctuate too much between high and low values within a short period of time to make such evaluations invalid. The framework allows service providers to specify the t_p based on preferences and practical experience.

3.5 Service Execution Engine

Our architecture adopts the concept of distributed service execution, where services could be executed on either the mobile device, the cloud or both. The service execution engine resides on the mobile device with a supporting remote execution module at the cloud side. The control of the service execution remains at the mobile device. The execution engine at the mobile provider may delegate the execution of a service partially or entirely on the cloud based on the recommendation of the offloading decision maker. Based on such a recommendation, the execution of a service might involve data transfer between the two parts of the execution engine.

3.6 Offloading Decision Maker

The offloading decision involves two aspects: 1) selecting the best available resource provider, and 2) determining the best execution plan for a service requests. Both aspects require full knowledge about the target operation and runtime context information.

3.6.1 Provider Selection

There are several parameters that make selecting where to offload computations a challenging decision. These parameters include the possibility of network failure, diversity of potential resource providers and mobility of both users and providers. Our observation is that dynamic mobile environments offer opportunities that could be considered when making offloading decisions. For example, a mobile device may use direct short-range communications to offload required computations to a nearby provider. This would avoid the long latency of the public cloud. In this section, we explore various possibilities of remote hosting and propose the *Follow-Me-Provider*, a dynamic resource provider selection scheme.

A mobile system can choose between multiple options for computation offloading:

- *Public cloud*. Mobile systems may use a public cloud to fulfil their offloading demands through creating a *clone image* or an *on-demand remote execution environment*. A clone image is a dedicated VM representing the mobile device on the cloud [10]. The device sends offloading requests to the clone device on a separate thread and results are integrated with the main execution thread on the mobile device. While this option incurs a fixed monetary cost for the continuously running clone VM, it avoids the overhead of creating and destroying VMs on-demand [33]. Data required for processing could be transferred to the clone device offline (i.e. before placing the offloading request) or online during execution. The decision whether to download required data online or offline depends on how frequently this data is used and how much does it cost to store the data on the cloud. The service provider must choose between storage cost and potential performance gain. In case of online data transfer, the clone device may download required data through a high speed interconnect with the data provider. This would significantly improve the overall performance and reduce the response time. Using a clone image, data could be cached for subsequent offloading requests, which results in more energy saving. The clone image is a better choice for mobile systems with heavy and frequent offloading demands. The second option is to create a remote execution environment (i.e. VMs, proxies, stubs) on-demand upon making offloading decisions [9]. Although this seems to be more realistic for users with low offloading demands, data transfer and management overhead of remote execution environments impact the overall latency and imply higher energy consumption.
- *Cloudlets*. A cloudlet is a hub that brings public cloud within close proximity of mobile users [30]. It can be viewed as the middle tier of a three tier architecture, mobile devices, cloudlets, and cloud infrastructures. Cloudlets can substitute the public cloud during network failures through offering essential services [34]. With the recent developments of smart vehicular technologies, capable mobile resource providers could also offer cloudlet services on the move.
- *Mobile cloud*. Mobile devices can collaborate to form a mobile cloud platform using virtualization techniques to offer resources on demand [2], [3]. This paradigm enables mobile devices to share their extra resources, mostly to offer location-based cloud-like services. While such an infrastructure might not be as powerful as public cloud, the close proximity to offloading entities improves communication latency. For example, a mobile user driving on a highway can take advantage of a platoon of smart vehicles moving along to execute resource-intensive tasks through a direct WiFi channel. Mobile cloud could be the only viable option for infrastructure-less environments, such as during disaster recovery or emergency situations.

To exploit the dynamicity of mobile environments, we propose the *Follow-Me-Provider* selection scheme as illustrated in Fig. 4. The *Follow-Me-Provider* employs the context of both the mobile system and runtime conditions to establish dynamic bindings between resource providers and computation offloading requests. The scheme utilizes the mobility of the offloading system and the amount of time required for computation to determine the best resource

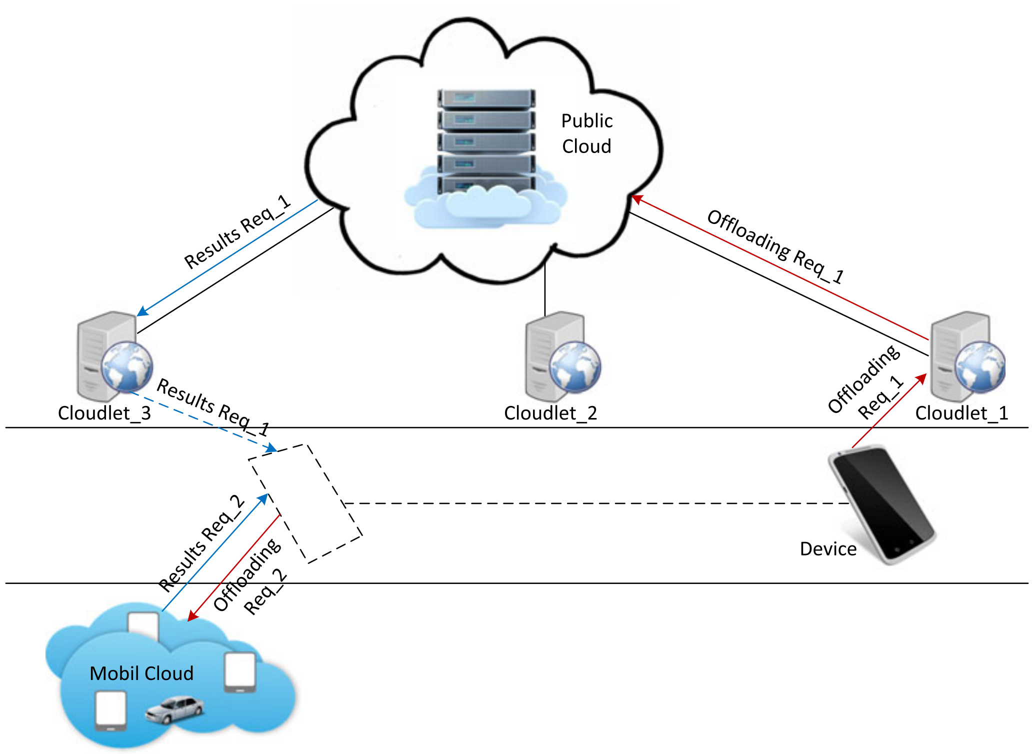

Fig. 4. Illustration of the *Follow-Me-Provider* selection scheme.

provider to carry out the computations and the resource provider that may forward the results. For example, in Fig. 4 the *offloading Req_1* is best to be offloaded to *cloudlet_1*. When computations are complete, the results are better to be sent via *cloudlet_3*. However, for *offloading Req_2* the mobile cloud in the nearby vicinity is the best choice, taking advantage of the direct connection and being in the communication range during required computation time. The *Follow-Me-Provider* also takes into account the data transfer requirements and the quality of the communication link with the data provider.

3.6.2 Plan Selection

The framework handles mobile services at the granularity of individual service operations, which are considered as the basic unit of computation that a service request may target. Assume that an operation (computing task) $\varepsilon = \mu + \delta$, where μ is the amount of computation that must be performed on the local system (i.e. requires access to local resources) and δ is the amount of computation that can be carried out remotely. Any of the two components may constitute the whole computing task when the other component is not applicable (e.g., $\delta = 0$ when the whole computing task requires access to a sensor in the mobile device). The remote component of the task δ requires m memory space and e amount of energy to execute. The computational speed of the mobile device is S_m (instructions/second), and S_{c,r_i} is the computational speed of a remote host server r_i. The execution may involve communications between various entities as shown in Fig. 1, where B is the link bandwidth and d_{in} and d_{out} are the sizes of data exchange between two entities. The mobile system consumes power (in watts), p_c for computing, p_i while idle, and p_t for transmitting data (sending or receiving). Although, in practice, sending data entails more energy consumption than receiving, for the purpose of this analysis, our model considers them identical.

The time t_m to process δ on the mobile system is:

$$t_m = r \times \frac{\delta}{S_m} + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_j}, \tag{1}$$

where r is the number of invocations and $n \leq 2$ indicating that the mobile system interacts with the user (service requester) over the link B_1 and *may* download required data from the data provider over the link B_2. If the task requires no data from the data provider, then $d_{in_2} = 0$ and $d_{out_2} = 0$. The first term in Eq. (1) represents the execution time on the mobile device and the second term indicates the data transmission time. We assume that the data required for multiple invocations is transferred only once.

If the task is offloaded, the time t_{c,r_i} to execute the task on a remote server r_i is:

$$t_{c,r_i} = r \times \frac{\delta}{S_{c,r_i}} + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_{j,r_i}} + t_\alpha, \tag{2}$$

where $n \leq 5$. The execution plan determines the value of n, indicating the possible interactions between various entities, as shown in Table 2. The time t_α represents any extra time required to build stubs or proxies in order to handle remote execution. Task offloading brings performance gain when $t_m \geq t_{c,r_i}$.

Similarly, the generic formula that calculates the energy consumption on mobile system is given by Eq. (3). This is a rough estimate of the energy consumption, but the model needs only to project relative values among various plans in order to determine the most energy-efficient plan

$$e_m = r \times \frac{\delta}{S_m} \times p_c + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_j} \times p_t, \qquad (3)$$

where $n = 1$ when the mobile system interacts only with user with no data requirements from the data provider, $n = 2$ otherwise. The first term in Eq. (3) represents the energy consumption by local computations, while the second term is the energy required for data transfer. When offloading takes place, the energy consumption on the mobile system is totally dependant on the offloading plan and can be generally calculated by:

$$e_{c,r_i} = k \times t_{c,r_i} \times p_i + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_{j,r_i}} \times p_t, \qquad (4)$$

where $n \leq 5$, $k = 0$ when *forward_response* is *'ON'* and $k = 1$ if it is *'OFF'*. It's worth mentioning that $k = 0$ also when asynchronous communication is used, however, there will be an overhead for re-establishing the connection. Offloading incurs energy saving when $e_m \geq e_{c,r_i}$.

Algorithm 1. Plan Selection Procedure.

```
Input: $\delta.profile$, $R$, $P$, $list\_lcxt$,
$list\_ecxt$, $\{m_{cr}, e_{cr}\}$
Output: $\{p, r\}$, //selected execution plan
  and provider
1  Function selectPlan($input\_list$)
2  //initialize holders of best plan and provider
3  $best\_p \leftarrow null$
4  $best\_r \leftarrow null$
5  $min\_t \leftarrow t_m$
6  if $e \leftarrow e_m < (e_{avail} - e_{cr})$ and $m < (m_{avail} - m_{cr})$ then
7    $best\_p \leftarrow p_1$ // local execution plan
8  end
9  foreach $r$ in $R$ do
10     foreach $p$ in $P$ do
11         if $m < (m_{avail} - m_{cr})$ then
12             $t \leftarrow t_c$
13             $e \leftarrow e_c$
14             if $t \leq min\_t$ and $e < (e_{avail} - e_{cr})$ then
15                 $min\_t \leftarrow t$
16                 $best\_p \leftarrow p$
17                 $best\_r \leftarrow r$
18             end
19         end
20     end
21 end
22 return $best\_p, best\_r$
```

The decision on whether to execute the service operations locally must ensure that the available resources on the mobile system satisfy the following constraints:

1. $m < m_{avail} - m_{cr}$
2. $e < e_{avail} - e_{cr}$

where m_{avail} indicates the available memory on the mobile device, e_{avail} indicates the remaining battery level, and both m_{cr} and e_{cr} are user-defined parameters based on preferences and context. These user-defined preferences are set to accommodate any special requirements, such as securing sufficient resources to maintain proper functionality of critical applications. The framework enables users to dynamically change these parameters according to their context.

Algorithm 1 illustrates the plan selection procedure for a particular computing task. The procedure excludes plans that do not satisfy local resource constraints and returns the plan that yields the smallest response time. Inputs of this procedure are:

- $\delta.profile$, the resource consumption profile of the computing task (*profiler*).
- $list\_lcxt = \{[B_1, B_2], S_m, p_c, p_i, p_t, m_{avail}, e_{avail}\}$, local context attributes of the mobile system (*context manager*).
- R, list of potential resource providers (*context manager*).
- P, possible offloading plans (*execution planner*).
- $list\_ecxt = \{S_c, [B_3, B_4, B_5], m_{avail}, e_{avail}\}$, the environment context for each potential resource provider $r \in R$ (*context manager*).
- $\{m_{cr}, e_{cr}\}$, user-defined constraints (*context manager*).

4 Implementation Details

We implement our validation prototype in Python. Python comes with a lightweight embedded HTTP server that is suitable for resource-constrained hosts, as well as many libraries that facilitate web service developments and deployments. We developed a RESTful web service that exposes multiple functionality as web service methods, each operation is represented with a unique URI in the form of http://base-address[host]/service-root/method-name. This service provides some image processing functionality ranging from low to high computational-intensity with various data transfer requirements, specifically *Blur*, *Blend*, *Steganography*, and *Tag*. The *Blur* operation blurs all identifiable objects in a certain image. The *Blend* operation blends two images gradually from left to right so that the left edge is purely from the first image and the right edge is purely from the second image. The *Steganography* operation implements a steganographic method to hide a text message inside an image. The *Tag* operation applies augmented reality techniques on an image taken by the device embedded camera. The image is labeled with the current location and the *Tag* operation annotate objects appear in the image, such as governmental buildings, tourist attractions, public services, business facilities, etc. Augmented reality is known as a computation-intensive process. This experimental setup represents a real case scenario of a photo sharing mobile service and is sufficient to illustrate the main aspects of our framework.

The web service is deployed on a Samsung I9100 Galaxy II (Dual-core 1.2 GHz Cortex-A9, 1 GB RAM) with a rooted Android 4.0.4 platform, connected to a WiFi network and is 3G-enabled. According to these specifications, $M = 2400$ MHz, $p_c = 0.9$, $p_i = 0.3$, and $p_t = 1.3$ all in Watts per second. The cloud provider is represented by an Amazon EC2 virtual machine of the type *'m1.large'* with an EC2 pre-configured image (AMI) of *'Ubuntu Server 12.04 LTS, 64 bit'*. We placed one image and the message-to-hide on the mobile device. Another image is placed on the cloud. The tagging

TABLE 1
Summary of the Experimental Data Placement

Data	Size	Location
Message to hide	150 KB	Mobile Device
Image 1	1.79 MB	Mobile Device
Image 2	1.84 MB	Cloud
Tagging Database	17 MB	Data Provider

TABLE 2
Possible Execution Plans for the Offered Service Operations

Plan	Exec. Location	Exec. Sequence
P1	M	$U \rightarrow M \rightarrow U$
P2	C	$U \rightarrow M \rightarrow C \rightarrow M \rightarrow U$
P3	C	$U \rightarrow M \rightarrow C \rightarrow U$
P4	M	$U \rightarrow M \rightarrow D \rightarrow M \rightarrow U$
P5	M&C	$U \rightarrow M \rightarrow C \rightarrow D \rightarrow C \rightarrow M \rightarrow U$
P6	M&C	$U \rightarrow M \rightarrow C \rightarrow D \rightarrow C \rightarrow U$

information database is hosted on a third-party data provider. This data placement allows us to test a variety of execution plans of various service requests. Table 1 illustrates the experimental setup, indicating where resources are located. We perform the experiments over a variety of wireless links with various levels of link quality between the mobile device, the client, data provider and the cloud.

According to this setup and based on data placements, possible execution plans for the service operations are shown in Table 2, where U represents the user, M represents the mobile device, C denotes the cloud, and D indicates the data provider. For example, P1 executes the required operation locally on the mobile resources, while P2 and P3 offload the execution to the cloud. However, in P3 the mobile system delegates the cloud to dispatch the response to the user directly. Not all plans are applicable for all operations, for example, $P4$ and $P5$ are applicable only for the *Tag* operation, where a third party data provider is involved in the processing. The arrows show the data transfer direction. In these experiments, we assume that communications between different parties is carried out in an asynchronous mode [35] to overcome the possibility of wireless link failures.

In our prototype, the profiler component uses several Python libraries to analyze the behaviour of service operations and to generate a behaviour and resource consumption profile for each operation. Guppy-PE [36] is a Python library and programming environment that provides memory sizing, profiling and analysis. It also includes a heap analysis toolset for memory related debugging. Guppy-PE provides the inter-function relations and shows the count of function calls. We also found that the *Memory Profiler* [37] Python library is efficient in the line-by-line analysis of memory consumption. The *Memory Profiler* tool exposes a number of APIs, such as *memory_usage*, that can be used by a third-party code to monitor the memory consumption. The *cProfile* [38] module is a built-in function that provides deterministic profiling for the CPU consumption of Python programs, from which our profiler determines the number of CPU cycles required for the execution of service

TABLE 3
Resource Consumptions of the Various Exposed Operations

Operation	CPU Time(m/c)	Mem. Usage (MB)
Blur	85/16	16.262
Blend	106/24	19.141
Steganography	146/40	12.363
Tag	4867/1236	54.253

operations. Table 3 shows the resource consumption of the various exposed operations by our web service.

The *Context Manager* uses *Iperf* [39] to monitor the link quality between different communicating entities. Iperf is a tool that measures the bandwidth performance of network links. Unfortunately, there is no accurate tool or commercial instrument that can measure the power consumption per instruction or individual processes. To date, the Android platform does not offer much with regard to energy consumption, but internal battery monitoring on a time-based level [40]. There are some recent research efforts towards this direction [41], [42], [43], but none of these efforts has offered a library accessible in the public domain yet. For rough estimates of power consumption per Android process, we use the Android open source project, *PowerProfile* [44]. This computation estimate is sufficient for comparison purposes between different execution plans. The context manager monitors the remaining battery power and keeps track of the consumption profile of other running applications. At the time a service request is received, the framework ensures that the energy constraint would not be violated by executing the service request whether locally or remotely on the cloud. However, preferences are given to energy efficient execution plans. Otherwise, the request is rejected.

5 Experimental Validation

In our experiments, processing is performed on the mobile, in the cloud, or both. Required data for processing may be located at any of the three locations, mobile device, cloud, and data provider. According to the required process and where the data is located, our mathematical model determines the best option to process the operation request based on the current context information and device resource constraints. Options include moving required data to the device or offloading the processing to the cloud with any required data from the device, the data provider, or both. To validate the model recommendation, we experimentally try all possible execution plans for a specific operation request and measure the end-to-end response time and energy consumption. The end-to-end response time includes communication, processing, and any overhead time to establish a network connection or generate remote execution proxies. Our expectation is that the experimental results should provide a strong backup of the model recommended option. In the normal practice, the mobile service execution environment uses the model to decide on the appropriate execution plan of a particular service request.

Table 4 shows an example of the actual average response time in contrast with the estimated response time and energy consumption of the various service operations.

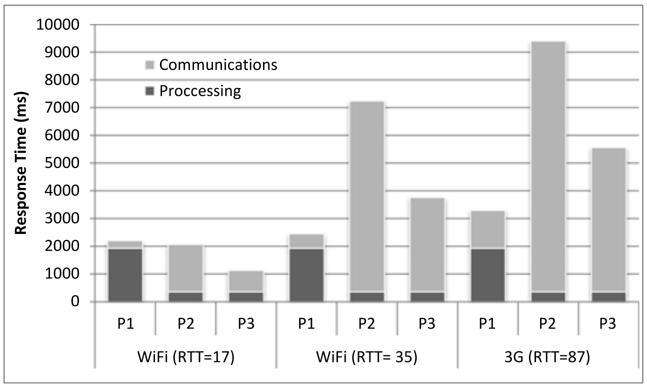

Fig. 5. Mean response time of the *Blur* operation.

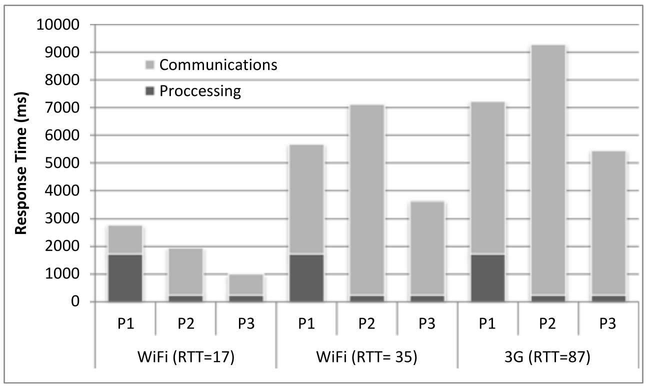

Fig. 6. Mean response time of the *Blend* operation.

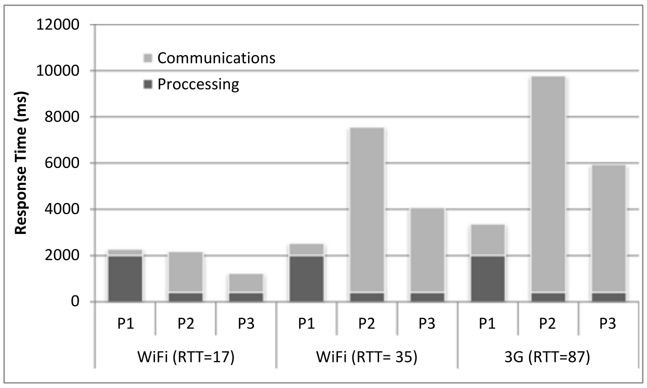

Fig. 7. Mean response time of the *Steganography* operation.

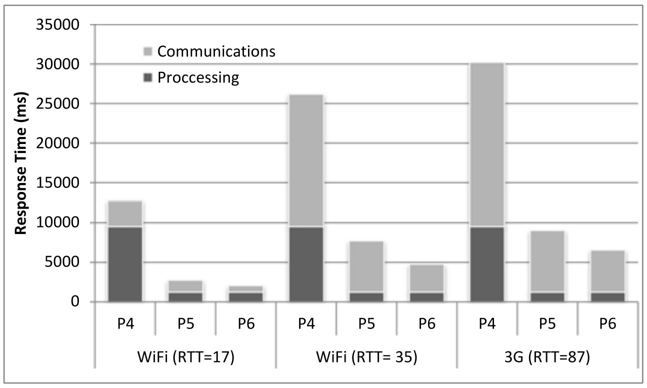

Fig. 8. Mean response time of the *Tag* operation.

TABLE 4
Actual Response Time in Contrast with Estimated Response Time and Energy Consumption of the Various Service Operations

Operation	Exec. Plan	Actual Res. Time	Estimated Res. time	Estimated Energy
Blur	P1	2205.39	2067.64	1.92
	P2	2074.27	2000.78	2.46
	<u>P3</u>	1131.78	1102.33	1.29
Blend	P1	2775.38	2556.58	2.63
	P2	1954.23	1737.74	2.16
	<u>P3</u>	1011.74	946.29	1.13
Steganography	P1	2276.73	2138.98	1.98
	P2	2177.38	1983.50	2.42
	<u>P3</u>	1234.87	1122.04	1.30
Tag	P4	12778.92	11142.56	10.68
	P5	2743.08	1964.33	2.06
	<u>P6</u>	2076.07	1510.62	1.47

Experimental results validate the suggested plan by our offloading model for each operation, which is underlined. Although there is a marginal difference between the actual response time and the estimated values, the offloading decision maker is able to select the plan that the yields a better response time while satisfying the resource constraints. We attribute this difference to the overhead time of generating proxies and stubs for remote execution as well as the delay incurred by the internal process of web servers. In addition, the advanced CPU technologies such as SpeedStep, Hyper-Threading, Pipelining, and Overclocking might also contribute to the deviation of the imperial values from calculated values with a certain offset. A system-specific calibration can capture such an offset and add it to the equation to make calculations accurate. However, estimates don't have to be strictly accurate since our model only needs to project relative differences among plans to select the proper one.

6 Experimental Results and Discussions

The performance of the framework varies significantly according to the several parameters including the data location, the link quality between different communicating entities and the selected execution plan. The context manager plays an important role through real time monitoring of resource consumption and network conditions on which the framework dynamically bases the choice of the the optimal execution plan. The experiments are performed under three different network conditions and settings: 1) the mobile device provider is connected through a fast WiFi link with an average Round Trip Time (RTT) = 17 ms while the client is wire connected, 2) both the mobile provider and the client are connected through a slow WiFi connectivity with an average RTT = 35 ms, and 3) both the provider and the client are connected over 3G with an average RTT=280 ms. In all settings the data service provider is linked to the cloud through a high speed interconnect with available bandwidth = 250.7 MB/s.

Fig. 5 shows the mean response time of the *Blur* operation with the possible execution plans in the three different settings. In this operation, the required data is located on

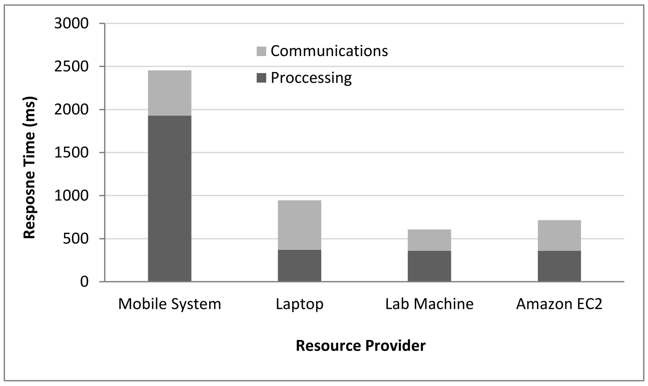

Fig. 9. The performance of different resource providers in the *Blur* operation.

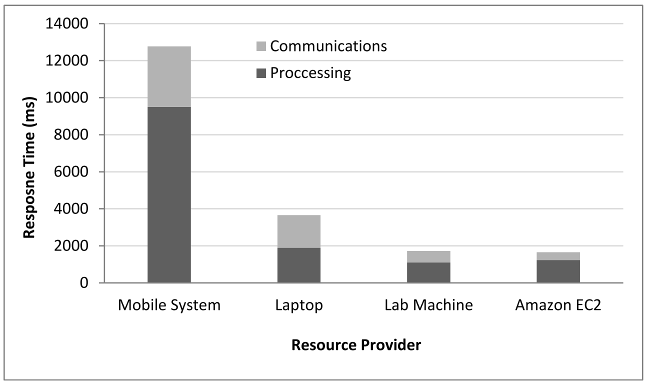

Fig. 10. The performance of different resource providers in the *Tag* operation.

TABLE 5
Potential Resource Providers for Computation Offloading

Attribute	r1	r2	r3
Description	Laptop	Lab Machine	Amazon EC2 *'m1.large'*
Architecture	64-bit	64-bit	64-bit
Support Type	Mobile Cloud	Cloudlet	Public Cloud
$S_c(GHz)$	2x2.4	2x2.8	2x2.8
$B_2(MB/S)$	32.0	10.5	6.3
$B_4(MB/S)$	14.5	45.5	250.8
$B_5(MB/S)$	3.2	20.3	20.3

the mobile device, which in our case has a little lower processing capability in contrast with the selected cloud server. This relatively small difference in processing capability between the mobile provider and the cloud does not give the cloud an edge for non-computational intensive processes, especially when communications take place over a low speed link. For the *Blur* operation request, only plan P3 with the case of high speed WiFi link (first setup) brings performance improvements. The difference between P3 and both P2 and P1 captures the speedup opportunity that the cloud may offer due to computational offloading. The experimental results reveal that P3 always yields better results than P2, while P1 might be a better choice when large data transfer is required, especially over slow interconnects. The *Steganography* operation demonstrates a similar behavior to the *Blur* request and is shown in Fig. 7. Since all the required data is available at the mobile provider and the operation is not computationally intensive, local execution proves to be more efficient except when data transfer is very fast, where offloading with P3 results in a faster response time and a lower energy consumption.

Fig. 6 illustrates the results of executing a blend operation request under the different settings. The execution of this service operation entails the transfer of one image to the other side, where processing occurs. In this case, offloading the computation to the cloud and allowing the cloud to dispatch the response to the user is always better. However, offloading achieves more than 3x overall speedup with high WiFi connectivity. We also observe that the P3 is the most energy efficient execution plan for the mobile provider.

The image tagging operation entails a large amount of data transfer from the data provider as well as the operation itself is computational-intensive. Resolving such a service request on a resource-constrained mobile provider significantly strains the limited resources and results in a high latency as shown in Fig. 8. Offloading such a request to the cloud improves the overall response time with orders of magnitude. For example, P6 achieves 6.1x, 5.5x and 4.6x speedup with settings 1, 2 and 3, respectively. The significant improvement can be attributed to the high speed interconnect between the cloud and the third-party data provider. In practice, the third-party data is most likely hosted on the cloud, which makes offloading a more viable option. The tagging operation also is an example of distributed execution, where part of the operation is performed on the mobile side, which is relating the image to a current user location, while the object recognition and labeling are performed on the cloud side.

The results highlight two main observations. First, offloading does not always guarantee better performance, especially when the process requires high data transfer over a low speed link. Second, offloading yields better performance when the cloud forwards the response to the user directly and is responsible for collecting the necessary data from the data cloud provider. The results also show that the option with the smallest response time is not always the choice of our model. For example, when the energy constraint that is set by the user could be compromised, the response time becomes of less concern. In fact, the user might resort to raising the critical energy threshold to secure sufficient energy for essential functionality or temporally critical applications, such as health care monitoring when the user is experiencing critical health conditions, or if the user is running a mobile-based navigation application while traveling. It is also worth noticing that P2 results in significant energy consumption due to high data transfer requirements back and forth to the cloud.

To study the impact of provider selection, we run the *Blur* (low data transfer requirement) and *Tag* (high data transfer requirements) operations on a number of resource providers with different settings. Table 5 shows the configuration and context information of these providers. In the experiments, we set the *forward_response* always *'ON'*, so that the remote execution environment can send the response to the user when offloading occurs.

Fig. 9 shows the performance of the various potential resource providers for the *Blur* operation. The lab machine (r2) outperforms the other resource providers due to the low latency connection with the user and low data transfer requirement from the mobile system. Although the laptop (r1) has a higher speed connection with the mobile system in contrast to other providers, the low quality connection with and user compromises the performance gain. If *'forward_response'* is turned *'OFF'*, the laptop option would be the most energy efficient and offer the best response time. Fig. 10 shows the performance of the same resource providers for the *Tag* operation. In this case, the public cloud (r3) brings the highest performance gain due to the high speed interconnect with the data provider. The other two options still perform significantly better than the mobile device but less efficient than the public cloud. The public cloud also continue to perform better even if the *'forward_response'* is turned *'OFF'*. From these results, we conclude that data transfer requirements and network conditions are significant attributes in making the offloading decision and selecting the most efficient resource provider.

7 Overhead Analysis

There are a number of elements of our framework that introduce overhead that may or may not directly impact the overall performance of computation offloading. In this section, we provide a subjective analysis of the various sources of overhead in our framework. We also show the framework's footprint and how much overhead does the operation of the framework incurs on mobile resources.

7.1 Context Management Overhead

Gathering and processing context information involve additional overhead on mobile systems. However, we argue that context information is a core component in current and future mobile applications (e.g. Siri, Viber, Google Now, etc.). Therefore, context management exists in mobile systems for adaptive and personalization purposes [45]. The overhead related to context management, both in energy consumption and processing time, is not specific to our framework. Furthermore, most of the context attributes are collected offline on a regular basis, incurring no extra delay on offloading decisions.

7.2 Decision Making Overhead

The time taken to make a decision results in a direct overhead on computation offloading. When a task can be offloaded to a remote server, the mobile system initiates the decision making procedure to decide whether offloading is beneficial. Making such a decision involves selecting the best available provider and the best execution plan (according to our offloading model). The time to select a resource provider is $O(k)$, where k is the number of available resource providers. Whereas the time to decide on the appropriate execution plan is $O(n)$, where n is the number of possible plans (in our framework, $n \leq 5$). The complexity of our decision making algorithm is $O(nk)$. Both k and n are relatively small and the delay overhead of decision making can be neglected in most cases. Based on our implementation experience, the decision making overhead is within a fraction of a millisecond.

TABLE 6
The Framework's Footprint in Terms of CPU Utilization, Memory Usage, Storage Space, and Average Battery Consumption of the Mobile Side Portion of the Framework in Both Idle and Active Modes

Running mode	CPU (%)	Memory (MB)	Storage (MB)	Battery Consumption (j/minute)
Idle mode	2.3	24.2	3.2	0.45
Active mode	11.6	65.6	3.2	1.2

7.3 Creating Remote Execution Environment

Creating a remote execution environment is another indirect overhead on computation offloading. The overhead varies according to the chosen offloading mechanism. Although the creation of this execution environment is carried out at the server side, a fraction of delay is added to the overall response time. The only exception is where mobile systems are represented on the cloud with a clone device [10]. Shiraz and Gani [33] study the diverse performance metrics associated with the deployment of VMs in cloud computing infrastructures. Based on our experience, we observe that instantiating a new VM instance on public cloud (specifically Amazon EC2) takes $30-60$ seconds. These numbers may be different for other cloud providers. In the case of RPC and RMI, such overhead is the same as if the computing task is invoked locally.

7.4 Framework Footprint

The framework operation adds extra overhead on mobile devices but in return offers robust computation offloading. The amount of such an overhead depends on the mode the framework is running on. The framework is in active mode while evaluating the offloading decision of a service request, whether offloading occurs or not. Otherwise, it runs in idle mode, where only the context manager is active in the background, gathering context information, and the request/response handler is listening for incoming service requests. To estimate the overhead that the framework poses on mobile resources, we measure its footprint in terms of CPU utilization, memory usage, storage space, and battery consumption in both active and idle modes. We remark that the measurement is carried out on the mobile device concerning the framework's footprint at the mobile service provider's side. This is to evaluate its feasibility on resource-constrained environments. Table 6 shows the framework's footprint on the mobile device. We observe that the battery consumption of the framework varies during the active mode according to the running activities and the cost of acquiring required context. The lower the cost of context acquisition, the lower the battery consumption of the framework while in active mode. In this experiment, we report the average battery consumption measured by the *PowerProfile* toolkit while the framework is running in active mode. The high CPU and memory footprint of the framework in the active mode is due to the evaluation of execution plans and the decision making process. Additionally, the context manager in the active mode acquires context information related to available mobile resource providers, which as well contribute to CPU utilization and battery consumption. In general, the overhead of our framework is much lower than regular context-aware mobile apps such as Google Now and Viber. For comparison purposes, the battery consumption of our framework in the idle mode is 20 percent lower than the WiFi scanner activity of the Android platform.

8 Conclusion

This paper presents a cloud-assisted mobile service framework. The objective is to augment the capabilities of mobile devices to become reliable service providers. The framework relies on a distributed service execution engine and a dynamic offloading scheme. Tasks that need to access local resources are executed on the mobile provider, other tasks could be offloaded to the cloud execution engine, if no constraints on remote execution exist. The framework includes a profiler, context manager, execution planner, offloading decision maker, and distributed service execution engine. The profiler characterizes the offered service operations and generates resource consumption profiles. The context manager gathers local and environment context information to make better decisions. The execution planner investigates possible execution plans based on locations of required data and current context information. The offloading decision maker selects the best execution plan and the most efficient resource provider based on our proposed *Follow-Me-Provider* scheme. The framework supports forwarding the response to the user from the remote execution environment (when applicable) to maximize the performance gain and reduce the energy consumption on the mobile system. We developed a prototype to validate the essential functionality of the framework and study the performance aspects. Experimental results demonstrate that the proposed cloud-assisted service framework offers significant latency improvements and incurs less energy consumption on the mobile system. Experiments also show that the framework operation has a lightweight footprint on mobile systems in contrast with regular mobile apps. We plan to extend the framework's functionality to support interrupted service execution due to connection failure. We also plan to enhance the *Follow-Me-Provider* selection scheme to determine cloud resource providers that offer trusted execution environments.

References

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

[2] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2009.

[3] K. Stølen, "Using infrastructure-less wireless networks to synchronize data among mobile devices: Extending ubishare," Master's thesis, Dept. Comput. Inform. Sci., Norwegian Univ. Sci. Technol., Trondheim, Norway, 2013.

[4] K. Elgazzar, P. Martin, and H. Hassanein, "A framework for efficient web services provisioning in mobile environments," in *Proc. 3rd Int. Conf. Mobile Comput., Appl., Services*, Oct. 2011, pp. 246–262.

[5] K. Elgazzar, H. S. Hassanein, and P. Martin, "Mobile web services: State of the art and challenges," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 3, pp. 173–188, Mar. 2014.

[6] T. Weerasinghe and I. Warren, "Empowering intermediary-based infrastructure for mobile service provisioning," in *Proc. IEEE Asia-Pacific Services Comput. Conf.*, 2009, pp. 414–421.

[7] M. Hassan, W. Zhao, and J. Yang, "Provisioning web services from resource constrained mobile devices," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 490–497.

[8] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *IEEE Commun. Mag.*, vol. 46, no. 1, pp. 56–63, Jan. 2008.

[9] 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 *Proc. 8th Int. Conf. Mobile Syst., Appl., Serv.*, 2010, pp. 49–62.

[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.

[11] S.-H. Hung, C.-S. Shih, J.-P. Shieh, C.-P. Lee, and Y.-H. Huang, "Executing mobile applications on the cloud: Framework and issues," *Comput. Math. Appl.*, vol. 63, no. 2, pp. 573–587, 2012.

[12] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proc. 12th Conf. Hot Topics Oper. Syst.*, 2009, pp. 1–5.

[13] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[14] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, 2012, pp. 21–28.

[15] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Feb. 2013.

[16] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proc. 1st ACM Workshop Mobile Cloud Comput. Services: Social Netw. Beyond*, 2010, pp. 6:1–6:5.

[17] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," in *Proc. ACM/IFIP/USENIX 10th Int. Conf. Middleware*, 2009, pp. 83–102.

[18] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proc. 2rd Int. Conf. Mobile Comput., Appl., Services*, 2010, pp. 59–79.

[19] K. Elgazzar, P. Martin, and H. S. Hassanein, "Empowering mobile service provisioning through cloud assistance," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Comput.*, 2013, pp. 9–12.

[20] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Machine learning-based runtime scheduler for mobile offloading framework," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Comput.*, 2013, pp. 17–25.

[21] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, 2002, pp. 217–226.

[22] M. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, Mar. 2010, pp. 217–226.

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

[24] K. Elgazzar, P. Martin, and H. Hassanein, "Enabling mobile web services provisioning," Queen's Univ., Kinston, ON K7L 3N6, Canada, Tech. Rep. 2012-598, Oct. 2012.

[25] H. J. La and S. D. Kim, "A conceptual framework for provisioning context-aware mobile cloud services," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 466–473.

[26] J. Ahn, J. Heo, S. Lim, and W. Kim, "A study on the application of patient location data for ubiquitous healthcare system based on LBS," in *Proc. 10th Int. Conf. Adv. Commun. Technol.*, 2008, vol. 3, pp. 2140–2143.

[27] C. Xin, "Location based service application in mobile phone serious game," in *Proc. Int. Joint Conf. Artif. Intell.*, Apr. 2009, pp. 50–52.

[28] J. Diaz, R. de A Maues, R. Soares, E. Nakamura, and C. Figueiredo, "Bluepass: An indoor bluetooth-based localization system for mobile applications," in *Proc. IEEE Symp. Comput. Commun.*, 2010, pp. 778–83.

[29] B. Altintas and T. Serif, "Indoor location detection with a rss-based short term memory technique (knn-stm)," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2012, pp. 794 –798.

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

[31] T. Jin, G. Noubir, and B. Sheng, "Wizi-cloud: Application-transparent dual zigbee-wifi radios for low power internet access," in *Proc. 30th Conf. Comput. Commun.*, Shanghai, China, 2011, pp. 1593–1601.

[32] Android SQLite [Online]. Available: http://developer.android.com/reference/android/database/sqlite/package-summary.html. Accessed: Jun. 2014.

[33] M. Shiraz and A. Gani, "Mobile cloud computing: Critical analysis of application deployment in virtual machines," in *Proc. Int'l Conf. Inform. Comput. Netw.*, 2012, pp. 11–16.

[34] M. Satyanarayanan, G. Lewis, E. Morris, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Comput.*, vol. 12, no. 4, pp. 40–49, Oct.–Dec. 2013.

[35] J. Fuller, M. Krishnan, K. Swenson, and J. Ricker. (2005, May 18). Oasis asynchronous service access protocol (asap) [Online]. Available: http://www.oasis-open.org/committees/documents.php?wg_abbrev=asap. Accessed: Jun. 2014

[36] Guppy-PE [Online]. Available: https://pypi.python.org/pypi/guppy/. Accessed: Jun. 2014.

[37] Memory Profiler [Online]. Available: https://pypi.python.org/pypi/memory_profiler. Accessed: Jun. 2014.

[38] The Python Profilers [Online]. Available: http://docs.python.org/2/library/profile.html. Accessed: Jun. 2014.

[39] Iperf [Online]. Available: https://code.google.com/p/iperf/. Accessed: Jun. 2014.

[40] Android Battery Monitoring [Online]. Available: http://developer.android.com/training/monitoring-device-state/battery-monitoring.html. Accessed: Jun. 2014.

[41] A. Rice and S. Hay, "Measuring mobile phone energy consumption for 802.11 wireless networkingy," *Pervasived Mobile Comput.*, vol. 6, no. 6, pp. 593–606, 2010.

[42] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application energy metering framework for android smartphones using kernel activity monitoring," in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 387–400.

[43] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2010, pp. 1–14.

[44] Andoid PowerProfile [Online]. Available: http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2_r1.1/com/android/internal/os/PowerProfile.java. Accessed: Jun. 2014.

[45] K. Elgazzar, H. S. Hassanein, and P. Martin, "Daas: Cloud-based mobile web service discovery," *Pervasive Mobile Comput.*, vol. 13, pp. 67–84, Aug. 2013.

Khalid Elgazzar received the PhD degree from Queen's University in 2013. He is a postdoctoral research fellow in the School of Computing at Queen's University. He has more than eight years of industrial experience in software design and development, and more than nine years of interdisciplinary academic teaching and research. His research interests span the areas of cloud computing, mobile and ubiquitous computing, context-aware systems, mobile services and applications, and elastic networking paradigms. He has received several recognitions and best paper awards at top international conferences. He leads a team on novel Ubiquitous cloud paradigms.

Patrick Martin received the BSc degree from the University of Toronto, the MSc degree from Queens University, and the PhD degree from the University of Toronto. He is a professor at the School of Computing at Queens University. He joined Queens University in 1984. He is also a visiting scientist with IBM's Centre for Advanced Studies. His research interests include database system performance, web services, autonomic computing systems, and cloud computing.

Hossam S. Hassanein is a leading authority in the areas of broadband, wireless and mobile networks architecture, protocols, control and performance evaluation. His record spans more than 400 publications in journals, conferences and book chapters, in addition to numerous keynotes and plenary talks in flagship venues. He has received several recognition and best papers awards at top international conferences. He is also the founder and director of the Telecommunications Research (TR) Lab at Queen's University School of Computing, with extensive international academic and industrial collaborations. He is an IEEE Communications Society Distinguished speaker (Distinguished lecturer 2008-2010). He is a senior member of the IEEE, and is a former chair of the IEEE Communication Society Technical Committee on Ad hoc and Sensor Networks (TC AHSN).