Empowering Mobile Service Provisioning Through Cloud Assistance

Khalid Elgazzar, Patrick Martin, Hossam S. Hassanein
School of Computing, Queen's University, Canada
{elgazzar, martin, hossam}@cs.queensu.ca

***Abstract*—The use of mobile devices as data service providers is on the rise. Mobile devices feature a large set of distinct characteristics that qualify them to be the most convenient computing platform for online services, both as consumers and providers. Mobile devices can take advantage of their mobility to provide location-based services and their association to a specific user to customize service offerings to fit personal preferences and current conditions. 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 data providers. Cloud computing offers mobile devices the opportunity to run resource-intensive tasks through computation offloading. The offloading decision is a tradeoff between data transfer and latency improvement to the benefit of alleviating the burden on mobile resource while improving the overall performance of service provisioning. This paper presents a framework for cloud-assisted mobile service provisioning, aimed at offering an augmented environment to resource-constrained mobile providers in order to deliver reliable services. The framework supports dynamic offloading based on the resource status at the mobile side and current network condition as well as user-defined energy constraints. It also enables the mobile provider to delegate the cloud to forward the service response directly to the user, given that no further processing is required by the provider. Performance evaluation shows up to 6x latency improvement for computational-intensive services that do not require large data transfer.**

***Index Terms*—Web service provisioning, mobile services, mobile computing, computation offloading**

I. Introduction

The role of mobile devices as data 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. Such lack of resource matching places limitations on the type of functionality and services that can be offered, restraining users from taking full advantage of their mobility passion. Cloud computing, therefore, offers the possibility to unleash the full potential of mobile devices to provide reliable data services.

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 might 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 not afford the energy requirements for such data transfer. In fact, the user might prefer to lower the bar of latency constraints to favor energy savings that might be needed for critical 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 Web services.

This research presents a distributed mobile Web service provisioning framework that reduces the burden on mobile resources through the offloading of resource-intensive processes. The framework takes advantage of cloud computing to bridge the gap between the limited resources of mobile environments and the growing resource demands of mobile service provisioning. An offloading decision model is proposed to determine whether or not remote execution of a resource request brings performance improvements. Based on this model the mobile service execution environment selects the best execution plan to resolve a service request according to the context of the requested Web resource and current network conditions.

The remainder of this paper is organized as follows. Section II outlines related work. Section III gives a brief distinction between Web services and mobile applications from the offloading perspective. Section IV presents the proposed cloud-assisted mobile service architecture. Implementation details and experimental validations are given in Section V and Section VI, respectively. Section VII presents the performance evaluation and offers a comprehensive discussion. Section VIII concludes the paper and draws future directions.

II. Related Work

Over the past few years, significant study has been done on the resource constraints of mobile devices as computing platforms. Computation offloading is one of the issues that has been extensively studied to enable such devices to run applications and services that require resources beyond what mobile devices can afford [1]. The offloading approach offers mobile devices the flexibility to customize the service interactions and optimize the resource consumption. However,

CFP13UCC-USB/13 $26.00 © 2013 IEEE
DOI 10.1109/UCC.2013.20

most of these efforts focus on the partitioning of mobile applications rather than Web service execution. For example, Giurgiu et al. [2] present a middleware that can distribute mobile applications between the mobile device and a remote server machine, 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. Similarly, CloneCloud [3] 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 on the cloud and communicates with the mobile device.

Sharing the same concern but from a different perspective, MAUI [4] enables energy-aware offloading of mobile code to a resource-rich computing infrastructure. MAUI aims to alleviate the burden on the limited energy resources of mobile devices while fulfilling the 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. MAUI uses such annotations to decide at runtime on the proper partitioning scheme.

Few research efforts have been dedicated to investigate the same issues from the perspective of Web service provisioning. Weerasinghe et al. [5] studied reliable mobile Web service provisioning with respect to availability and scalability. The authors propose a proxy-based middleware to bootstrap the performance of mobile Web 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. [6] present a distributed mobile service provisioning framework that partitions the execution of resource-intensive Web 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 partition technique relies solely on the available resources. If the available resources satisfy the Web services execution requirements, the execution is performed entirely on the mobile side. In contrast, our framework selects the best execution plan with the minimum response time and energy efficiency, while satisfying the resource constraints with respect to both execution requirements and user preferences.

III. Mobile Services vs. Applications: A Partitioning Perspective

Partitioning mobile applications between the mobile device and cloud computing saves scarce mobile resources, while leveraging the performance of mobile applications [2]–[4]. Several research efforts have contributed mechanisms and algorithms on how to optimally split an application to achieve a specific objective, such as saving energy or bandwidth, or to execute within a particular time. In contrast, Web services are self-contained and self-described to support interoperability between heterogeneous platforms with standard interfaces. Web service internal operations (in SOAP these are called methods while in RESTful Web services, they are called resources) are typically loosely-coupled, which means they are, in most cases, independent of each other. Bonds between these operations are weak, if they exist at all. Hence, a user request may invoke a particular method that independently performs the required functionality. In such a case, partitioning the execution of a single service operation is unlikely to bring performance benefits. This results in executing independent service operations entirely on a single side. Therefore, our proposed framework pays little attention to partitioning of single operation execution. However, in cases where the execution encompasses multiple functionality, the framework offers offloading of functionality that does not require access to local resources.

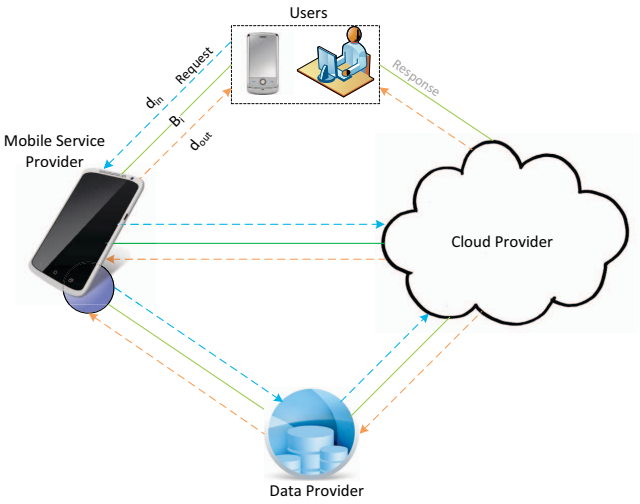

Fig. 1: An abstract view of cloud-assisted mobile Web service architecture.

IV. Cloud-assisted Mobile Service Architecture

This paper presents a cloud-assisted framework for provisioning Web services 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 Figure 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

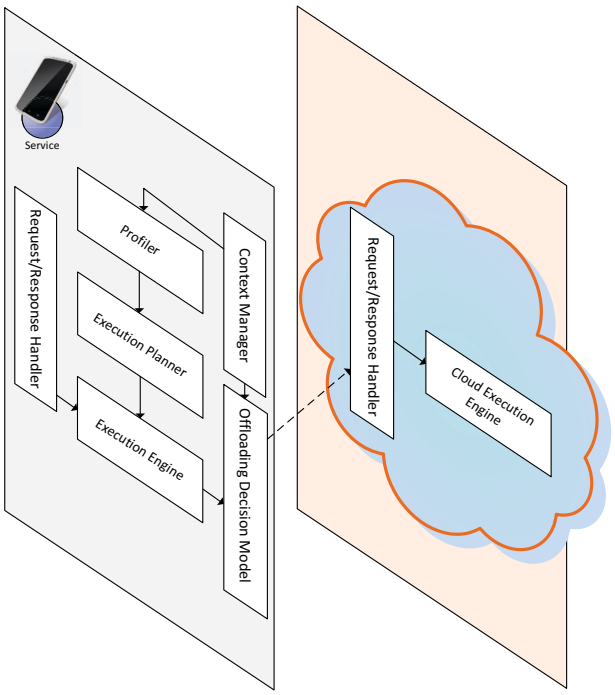

Fig. 2: An abstract view of cloud-assisted mobile Web service architecture.

tasks. Web service operations might 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 storage provider.

In this architecture, 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 user. It is also possible that the provider may 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 Module*. Figure 2 depicts an abstract view of the overall framework architecture. The functionality of each component is discussed in the following with the major focus being the offloading decision module.

A. Request/Response Handler

Internet users can request access to Web services or Web contents. A Web service request could be SOAP/XML for SOAP-based Web services [7] or an HTTP request for REST-based Web services [7]. By design, RESTful requests point directly to specific operations that carry out the required functionality whereas SOAP requests imply functionally based on associated 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 Web 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, whereas HTTP requests for Web services are analyzed directly by a servlet that selects the appropriate Web service operation to respond to these requests based on the class and method annotations.

B. Profiler

This component is responsible for analyzing the characteristics of various Web 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 Web 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 stand alone function. The profiler runs Web 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 dependency and inter-relations between one another. The profiler then generates a resource consumption profile for each Web service with a separate section for each operation as shown in Listing 1. More details about different types of profilers and profiling strategies can be found in [3, 4].

The planner module uses the information in the consumption profile to generate possible execution plans. The offloading decision module uses the execution plans along with the context information collected by the context manager to select the best execution strategy for a specific Web service operation (method) request.

C. Context Manager

The context manager gathers information about the link quality between different entities and available bandwidth. It also monitors resource availability on the mobile provider side, including CPU utilization, available memory, remaining battery life, and running applications. The context information and Web service consumption profiles are used by the offloading decision model to calculate the optimal execution plan that fits the device constraints and user preferences.

D. Execution Planner

The execution planner determines the various possible execution plans for each Web operation based on available information about each operation and the behavior profile generated from the profiler. Each operation can be executed in a variety of different ways.

Listing 1: A snippet of a resource consumption profile

```
<?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:lenergy>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>
```

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 the two. Service developers may specify that particular operations are to be strictly executed on mobile devices due to security reasons or privacy concerns such as the case when a provider wishes to ensure full privacy of its customers' information. Although current service description standards do not support such a feature, a recent initiative has been proposed [6] on how to include such requirements in the service description. Execution plans are generated offline to reduce the resource contention overhead at runtime.

The planner starts with the possibility of performing the execution locally on the mobile device, where the device acquires all the required data for processing and sends back a response to the user. If there are no specific requirements for local resource access, the planner generates a plan consisting entirely of remote processing. When the remote processing option is considered, the planner checks the possibility whether the response can be sent to the user from the remote location directly. The framework supports this feature, given that no processing is required at the mobile side. This reduces the communication and the consumption of mobile resources and improves the overall response time. If the provider does not want to share the customer's information, the possibility that the cloud forwards the response to users is no longer valid, regardless of potential performance benefits. If the required operation encompasses independent functions that can be performed separately, further plans of partitioning are considered. Several partitioning strategies are discussed in [2]–[4].

A plan evaluation is performed at runtime once an invocation request is received at the mobile provider's side. Since the churn of Web 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 a short interval. It's uncommon that network conditions (bandwidth BW 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 empirical experience. Execution plans may involve local, remote, or hybrid execution through partitioning.

E. 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 module. Based on such a recommendation, the execution of a service might involve data transfer between the two parts of the execution engine.

F. Offloading Decision Module

The offloading decision model provides the service execution engine with the best option to resolve a Web service request based on the possible execution plans of the target operation and runtime context information.

The framework handles mobile Web services at the granularity of individual Web service operations, which are considered as the basic unit of computation that a service request may target. Assume that an operation requires c computing instructions, m memory space, and amount of energy e to execute. The speed of the mobile device is M (instructions/second), and S is the speed of a cloud host server. The execution of a Web service operation may involve communications between some or all of the entities shown in Figure 2, where B is the link bandwidth and d_{in} and d_{out} are the data size exchanged between two entities, respectively. 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 overall response is calculated by the generic formula shown in Eq. (1) as follows:

$$RT = \frac{C_m}{M} + \frac{C_c}{S} + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_j} + t_\alpha \qquad (1)$$

The equation encompasses three main terms. The first and second terms represent the execution time on mobile device and the cloud, respectively, where C_m is the execution cycles carried out by mobile device, n is number of links in a plan, C_c is the execution cycles carried by cloud, and $C = C_m + C_c$. The third term indicates the data transmission time between the various involving entities and t_α represents any extra time

required to build stubs or proxies in order to handle remote execution. The maximum possible links between all entities is $n = 5$, however, n varies according the active links in a particular execution plan. For example, if only the mobile provider and a user are solely communicating throughout the course of the Web service execution, then $n = 1$, indicating the link between the user and mobile device, $C_m = C$, and $C_c = 0$ indicates that no execution occurs on the cloud. The cloud server speed S can be expressed as a multiple of M where $S = f \times M$. Eq. (1) can then be rewritten as follows.

$$RT = \frac{1}{M} \times \left(C_m + \frac{C_c}{f} \right) + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_j} + t_\alpha \quad (2)$$

Similarly, the generic formula that calculates the energy consumption at the mobile provider's side is calculated by Eq. (3).

$$E = \frac{C_m}{M} \times p_c + \frac{C_c}{f \times M} \times p_i + \sum_{j=1}^{n} \frac{d_{in_j} + d_{out_j}}{B_j} \times p_t \quad (3)$$

The decision on whether to execute the Web service operations locally must ensure that available resources of the mobile device 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. Such 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.

V. Implementation Details

We implemented 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 have 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 Web 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 labels identifiable objects that appear in an image taken by the device embedded camera. This image is then flagged with the current location and the *Tag* operation matches objects appear in the image, such as governmental buildings, tourist attractions, public services, business facilities, etc., with stored objects and tags them. This operation is known as augmented reality and is a resource-intensive process.

TABLE I: 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	Storage Provider

TABLE II: Possible execution plans for the offered Web 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	$U \rightarrow M \rightarrow C \rightarrow D \rightarrow C \rightarrow U$

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 side 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 information database is hosted on a third-party data storage provider. This data placement allows us to test a variety of execution plans of various service requests. Table I 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 storage provider and the cloud.

According to this setup and based on data placements, possible execution plans for the Web service operations are shown in Table II, where U represents the user, M denotes Mobile, 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 provider delegates the cloud to dispatch the response to the user directly, whenever applicable. 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 [8] to overcome and possibility of wireless link failures.

In our prototype, the profiler component uses several python libraries to analyze the behaviour of Web service operations

TABLE III: 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

and to generate a behaviour and resource consumption profile for each operation. Guppy-PE [9] 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* [10] python library is efficient in the line-by-line analysis of memory consumption. Memory Profiler 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* [11] 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 Web service operations. Table III shows the resource consumption of the various exposed operations by our Web service.

The *Context Manager* uses *Iperf* [12] 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 [13]. There are some recent research efforts towards this direction [14]–[16], but none of these efforts has offered a library accessible in the public domain yet. For rough estimations about power consumption per Android process, we use the Android open source project, *PowerProfile* [17]. However, the power consumption estimate in our prototype is based solely on computations. 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.

VI. Experimental Validation

In our experiments, processing is performed either on the mobile device or in the cloud. Required data for processing may be located at any of the three locations, mobile device, cloud, and data storage 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 storage, 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 Web service execution environment uses the model to decide on the appropriate execution plan of a particular service request.

TABLE IV: Actual response time in contrast with estimated response time and energy consumption of the various Web service operations.

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

Table IV shows an example of the actual average response time in contrast with the estimated response time and energy consumption of the various Web service operations. The recommended execution plan by our model for each operation is underlined. Although there is a marginal difference between the actual response time and the estimated values, the offloading decision module 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 in order to select the proper one.

VII. Experimental Results & Discussions

The performance of the framework varies significantly according to the several parameters including the data location,

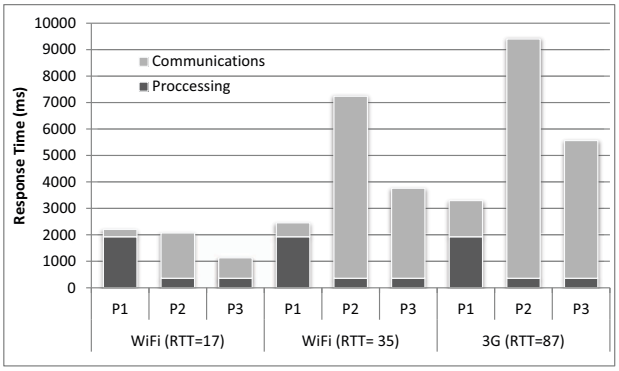

Fig. 3: The mean response time of the *Blur* operation.

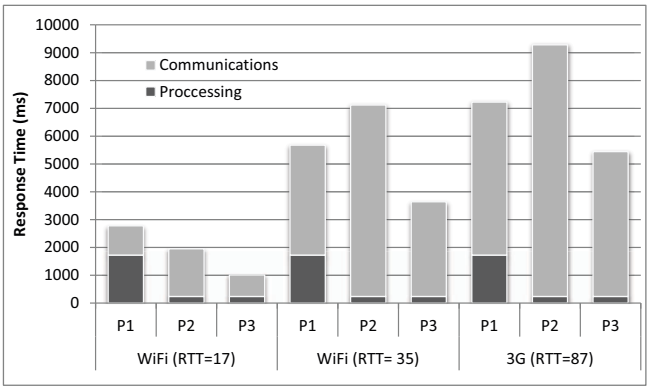

Fig. 4: The mean response time of the *Blend* operation.

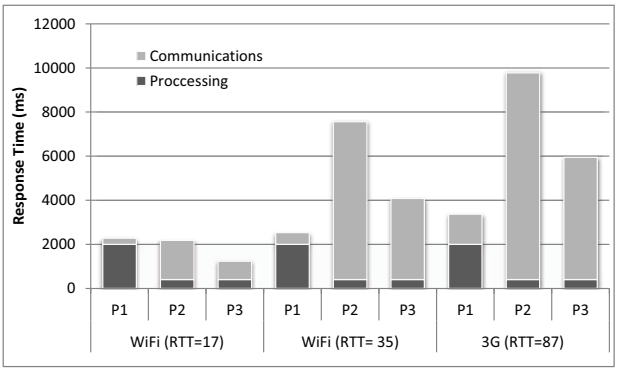

Fig. 5: The mean response time of the *Steganography* operation.

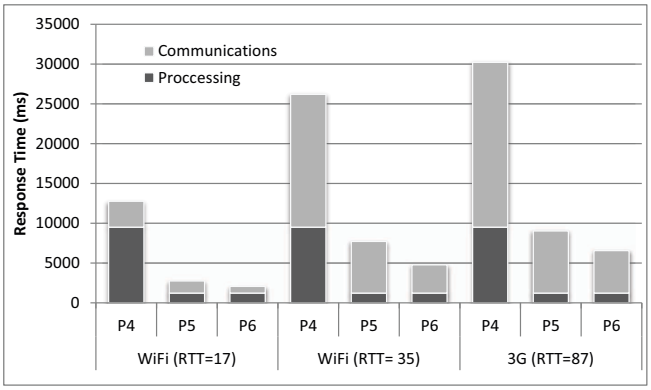

Fig. 6: The mean response time of the *Tag* operation.

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.

Figure 3 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 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 (1^{st} 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 Figure 5. 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.

Figure 4 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 storage 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 Figure 6. 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 to the cloud 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 least 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 important 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.

VIII. Conclusion

This paper presents a distributed mobile Web service provisioning 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 technique. Tasks that need to access local resources are executed on the mobile provider, while the rest could be offloaded to the cloud execution engine, if no constraints on remote execution exist. The framework includes a profiler and an execution planner. The profiler characterizes the offered Web service operations and generates resource consumption profiles. The execution planner investigates all possible execution plans based on locations of required data and current context information. The service execution engine evaluates these plans and selects the best resource-efficient plan that, in addition to satisfying the resource constraints, yields better performance and lower latency. 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 provisioning framework offers significant latency improvements and less energy consumption at the mobile provider's side. We plan to extend the framework functionality to support interrupted service execution and enable the communication over a variety of wireless network technologies.

References

[1] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the 3rd ACM workshop on Mobile Cloud Computing and Services*, MCS '12, (New York, NY, USA), pp. 21–28, ACM, 2012.

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

[3] 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*, (New York, NY, USA), pp. 301–314, ACM, 2011.

[4] 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*, pp. 49–62, 2010.

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

[6] M. Hassan, W. Zhao, and J. Yang, "Provisioning web services from resource constrained mobile devices," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, pp. 490–497, IEEE Computer Society, 2010.

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

[8] J. Fuller, M. Krishnan, K. Swenson, and J. Ricker, "Oasis asynchronous service access protocol (asap)," May 18 2005. http://www.oasis-open.org/committees/documents.php?wg_abbrev=asap. [Accessed: Jul. 9, 2013].

[9] Guppy-PE, https://pypi.python.org/pypi/guppy/. [Accessed: Jul. 9, 2013].

[10] Memory Profiler, https://pypi.python.org/pypi/memory_profiler. [Accessed: Jul. 9, 2013].

[11] The Python Profilers, http://docs.python.org/2/library/profile.html. [Accessed: Jul. 9, 2013].

[12] Iperf, https://code.google.com/p/iperf/. [Accessed: Jul. 9, 2013].

[13] Android Battery Monitoring, http://developer.android.com/training/monitoring-device-state/battery-monitoring.html. [Accessed: Jul. 9, 2013].

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

[15] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphones using kernel activity monitoring," in *USENIX Annual Technical Conference*, 2012.

[16] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, (Berkeley, CA, USA), pp. 21–21, USENIX Association, 2010.

[17] Andoid PowerProfile, http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/2.2_r1.1/com/android/internal/os/PowerProfile.java. [Accessed: Jul. 9, 2013].