Context-Aware Mobile Application Task Offloading to the Cloud

Hanan Elazhary
Computer Science Department
Faculty of Computing & Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia
Computers and Systems Department
Electronics Research Institute, Cairo, Egypt

Saja Aloraini and Roa'a Aljuraid
Computer Science Department
Faculty of Computing & Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia

***Abstract*—One of the benefits of mobile cloud computing is the ability to offload mobile applications to the cloud for many reasons including performance enhancement and reduced resource consumption. This paper is concerned with offloading of context-aware mobile applications, in which actions or tasks are executed in certain contexts and offloading those tasks needs to be itself context-aware to be advantageous. The paper investigates candidate techniques and development models in the literature to identify suitable ones. Accordingly, the paper proposes the practical Context-Aware Mobile applications Offloading (CAMO) development model, which we developed in Java for the Android platform. Programmers can exploit the independency of the tasks of a typical context-aware mobile application and use CAMO to profile each task in isolation on the mobile and the cloud. The paper introduces the concept of a task-offloading plan in which programmers specify a criterion and/or an objective for offloading a task in a specific context. Offloading criteria allow rapid offloading in case the mobile environment does not change frequently. Based on the profiling results, programmers can use the classes and methods of CAMO to develop one or more custom offloading plans for each task or use pre-specified plans, criterion and objectives. We provide three example tasks with details of their profiling and analysis for developing corresponding offloading plans. CAMO is general and flexible enough for offloading any application partitioned into independent modules. Empirical evaluation shows extreme satisfaction of mobile application developers with its capabilities.**

Keywords*—Application offloading; Context awareness; Distributed systems; Mobile application; Mobile cloud computing

I. INTRODUCTION

Context-aware mobile computing refers to developing mobile applications whose behaviour depends on context. In the broad sense, we can define context as the state of the mobile, the application or the user [1]. Context is extracted through internal mobile sensors and hardware features in addition to other external sources. Such raw context is typically, processed to a higher-level more understandable form [2]. For example, the current readings of the GPS system can be converted into current *city*, *country*, or *continent*. An example of a context-aware mobile application is an application that reduces the brightness of the screen in bright light and vice versa.

Mobile cloud computing [3] refers to mobile computing that exploits the theoretically infinite cloud resources to make up for the limited mobile phone resources. One approach involves saving data and especially Big Data on the cloud [4, 5, 6]. Another approach involves offloading execution of a mobile application to the cloud [7]. Researchers have proposed many techniques, frameworks, and development models for this purpose. Nevertheless, they mainly consider large applications that require partitioning to determine partitions they should offload or migrate to the cloud in order to resume execution.

This paper is concerned with offloading of context-aware mobile applications. A typical context-aware mobile application is essentially partitioned into local partitions that are responsible for checking the mobile context and tasks that take place when specific contexts are satisfied. Such tasks may run locally or remotely according to their requirements. For example, tasks responsible for mobile adaptations should run locally. Typically, those tasks are independent and start from scratch when their corresponding contexts are satisfied. This implies that complex offloading techniques might not be necessary and that we can employ ones that are more efficient. Towards our goal, we summarise the contributions of the paper as follows:

- To the best of our knowledge, this paper is the first to consider and study offloading of typical context-aware mobile applications in a context-aware fashion.
- We provide a thoroughly investigation of different techniques, frameworks, and development models proposed in the literature to examine their suitability for those applications.
- We propose (and developed) the Context-Aware Mobile applications Offloading (CAMO) development model in Java for the Android platform with several classes and methods that help programmers in developing off-loadable context-aware mobile applications. For example, programmers can exploit the independency of the tasks of such applications and use CAMO to profile each task in isolation on the mobile and the cloud.

- We propose the concept of task offloading plans, where programmers can use profiling results to determine conditions under which CAMO would offload each task, rather than merely whether it should offload the task as in most research studies.
- Using CAMO, the programmers can develop custom task-offloading plans, in which an offloading criterion (such as maximum or minimum data size) allows rapid offloading in case the mobile environment does not frequently change. The programmer can also specify an objective that should be satisfied by the criterion (such as maximum delay) and how to update the criterion otherwise. This opposes similar research studies in the literature that rely merely on the objective for making the offloading decision.
- CAMO provides pre-specified task-offloading plans, criteria, and objectives to the programmers to help them in developing their off-loadable applications.
- CAMO is general enough to be used for any mobile application partitioned into independent modules.

The following section provides a thorough investigation of mobile application offloading techniques, frameworks and development models in addition to corresponding challenges. It also discusses requirements of context-aware mobile application offloading. This is followed by a description of the details of the proposed development model, CAMO. Experiments based on CAMO are then provided accompanied by discussion. The last two sections provide results of the empirical evaluation of CAMO and the conclusion and directions for future research.

II. Mobile Application Offloading

Mobile application offloading to the cloud has drawn researchers' attention due to the limited resources of smartphones (such as computing power, memory size, storage capacity and battery capacity) and the virtually infinite resources offered by the cloud. We can broadly classify application-offloading techniques into partitioning techniques, migration techniques and replay techniques, possibly coupled with context-awareness. Some research studies merely propose offloading techniques while others propose development models and frameworks for off-loadable mobile applications.

A. Partitioning Techniques

Partitioning techniques are concerned with how to partition an application and how to decide which of the resulting partitions should run locally (such as partitions that require user interactions and mobile interactions to obtain GPS information for example) and which should run remotely (such as resource-intensive partitions) on which cloud. We can broadly classify partitioning techniques into graph-based, linear programming-based and annotation-based techniques [7].

Graph-based techniques represent the parameters of a given mobile application using a graph and seek to partition the application and decide which partitions would be offloaded to which clouds [8, 9, 10]. For example, in CloneCloud [9], a graph represents the modules of the mobile application. An *analyser* is responsible for determining possible methods to partition the graph representation of the application between the mobile and the cloud. A *profiler* generates a cost model for the application (in terms of execution time and power consumption), under different possible partitioning methods via executions on both the mobile and the clone cloud with a random set of inputs. Finally, the *optimisation solver* determines the best partitioning method (among those generated by the analyser) that optimises an objective function (using the cost model generated by the profiler) to be used at runtime. It is worth noting that the graph-partitioning problem is Non-deterministic Polynomial Complete (NPC) and so most efficient techniques require manual annotations to the application by the developers to provide cues to guide the partitioning process [7].

Linear-programming based techniques [11, 12] on the other hand, represent the partitioning problem as a mathematical optimisation problem and use linear programming methods to optimise application partitioning. For example, the Mobile Augmentation Cloud Services (MACS) middleware [11] assumes an application is partitioned into modules. A cost function is defined in terms of the computing cost and the memory cost of each application module on the mobile and its transmission cost to the cloud in addition to a Boolean variable indicating whether it would be offloaded. Each of the three above costs is given a corresponding weight, and linear programming methods are used to optimise the cost function to determine whether to offload each module.

Some research studies in the literature [13, 14] combine features from both graph-based techniques and linear programming-based techniques. For example, Sinha and Kulkarni [14] proposed representing the mobile application environment (such as the available clouds, the computing powers, the memory sizes and the communication links capacities) using a graph and then using linear programming methods to determine how to partition the application such that an objective cost function is minimised.

Annotation-based techniques such as Cyber Foraging [15, 16] and J-Orchestra [17] require extensive annotations to the mobile applications by the developers to guide the partitioning process using alternative methods. For example, in Cyber Foraging [15, 16], a language called *Vivendi* is used by the developer to describe the fidelity and tactics of a mobile application. The fidelity of an application is a normalised measure of its quality expressed as a number between zero and one, while the tactics are the possible partitions of the application. Finally, a *Chroma* scans the available tactics and selects the best partitioning plan that maximises the ratio between the application fidelity and latency.

It is clear that a major problem with such techniques is that they require either manual annotations or complex representations, which may be hindering to most mobile application programmers. Fortunately, as previously noted, a typical context-aware mobile application is inherently partitioned and so each task can be processed in isolation to

determine conditions under which it would be offloaded rather than merely whether it should be offloaded.

B. Migration Techniques

Migration techniques are used to migrating processes and virtual machines over networks. Examples of such techniques include the Zap system [18] in which a PrOcess Domain (*pod*) is a virtual machine with a virtual operating system view encapsulating a group of processes allowing them to migrate between machines running different operating systems to resume execution remotely. In the Internet Suspend/Resume (ISR) system [19, 20], a *parcel* is a virtual machine that encapsulates user-specified operating system settings, applications, and documents such that the parcel can be suspended before migration and resumed after migration.

Live virtual machine migration has been extensively studied for cloud data centres by constantly conveying changes from one virtual machine to another [21]. It is worth noting that live virtual machine migration across a WAN is more complicated due to inherent challenges such as long latency and variable or restricted bandwidth [22].

It is clear that such techniques are not suitable for typical context-aware mobile applications that start tasks from scratch when corresponding contexts are satisfied.

C. Replay Techniques

The idea of replay techniques is to record an execution of an application such that it can be later replayed. Deterministic replay refers to replay that can be fully reproduced deterministically. It has been shown to be useful for many purposes such as fault tolerance [23], workload execution tracing [24] and debugging [25]. Deterministic replay has also been proposed for *coarse-grained* mobile application cloud offloading without partitioning [26].

Non-deterministic replay, on the other hand, includes inputs such as a keyboard input or a camera input. The opportunistic replay technique [27] has been proposed to reduce the overhead associated with virtual machine migration by recording the non-deterministic events of user interactions with the application via the keyboard or the mouse and using the resulting interaction log to replay the application on the cloud. Hung et al. [28] extended this idea by proposing a framework in which programmers insert pseudo checkpoints in an application to mark locations at which the application can resume whenever it is paused. At each pseudo checkpoint, the input events are recorded, and on pause, the state of the application and the events are saved such that the application can be resumed starting from the nearest pseudo checkpoint and can be replayed using the recorded events until it reaches the state at which it was paused. The authors proposed using this technique to offload mobile applications to the cloud on a virtual machine that is as close as possible to the mobile environment provided that the machine holds a copy of both the application and the corresponding data. Data can be synchronised only upon the application request.

It is clear that such techniques, like migration techniques, are not suitable for typical context-aware mobile applications that start tasks from scratch when their contexts are satisfied.

D. Context Awareness

Context-awareness has been associated with mobile application cloud offloading in many research studies to make informed dynamic offloading decisions at runtime since offloading is not always advantageous [29]. For example, Zhou et al. [30] proposed a technique that enables making dynamic offloading decisions of an independent mobile application process at run time by selecting a suitable wireless channel and cloud resources that satisfy a set of quality-of-service (QoS) requirements while minimising cost and energy consumption. Cuervo et al. [31], on the other hand, proposed the MAUI system that requires code annotation by the programmer specifying off-loadable methods or classes. At runtime, it represents the offloading problem as a linear programming problem based on the CPU cost, the communication cost (size of the method state) and the bandwidth and latency of the available channels and solves the problem by making an offloading decision that maximises energy saving. Ellouze et al. [32] proposed another technique for making dynamic offloading decisions of independent mobile application processes at runtime based on the CPU load and battery state unless the offloading process itself is energy inefficient or violates the user Quality of Experience (QoE). Possible context measures include:

- The available resources (such as computing power, memory size, storage capacity and battery capacity) on the mobile versus the available cloud resources
- The execution time on the mobile versus the execution time on the cloud in addition to the offloading time
- Local computing energy consumption versus offloading energy consumption
- The specifications of the application and its objectives and the satisfaction of its QoS and QoE requirements on the mobile versus on the cloud

It is clear that the above techniques are more or less mobile application partitioning techniques except that the decision is made at runtime. In other words, they share the requirement of complex representations and/or code annotations as well as energy and storage space consumption and additional overhead at runtime in order to make such excessive computations. As previously noted CAMO considers this by allowing rapid offloading based on criteria rather than based on objectives in case the mobile environment does not frequently change.

E. Development Models and Frameworks

Some development models and frameworks have been proposed for helping developers in integrating offloading capability with mobile applications. For example, Zhang et al. [33] developed an SDK that can be used by developers to build

weblets by extending a corresponding abstract class. A weblet is an application partition that can change in state to be running, paused (before migration) or resumed (after migration). Application partitioning and the remaining functionalities are left to the developer. Unfortunately, this SDK is only suitable for migrating weblets. The Cuckoo framework [34] helps developers in building smartphone applications that can be offloaded dynamically at runtime. The developers have to identify the compute intensive code partitions and write both local and remote code implementation. Nevertheless, this was merely a prototype that employed very simple heuristics to make the offloading decisions. The µCloud [35] is another SDK intended to help developers in developing Java components that can be executed by a cloud orchestrator or a mobile orchestrator, but the developer has to partition the code to identify the different components and on which orchestrator each should be executed. The Uniport framework [36] is intended for developing applications that can run remotely based on the Model-View-Controller (MVC) architecture, but it only considers network availability as the trigger for remote execution of a replica of the application. The MobiByte system [37] allows developers to specify objectives for offloading each of the partitions of a given mobile application. Such objectives include performance enhancement, energy efficiency or merely execution under scarce resources. Nevertheless, it requires examining offloading objectives at runtime even if the mobile environment does not change frequently. Besides, it is not flexible enough to allow developing custom offloading plans or more than one offloading plan for a single partition in different contexts. CAMO addresses those drawbacks.

F. Challenges of Mobile Application Cloud Offloading

Many challenges face the success and adoption of mobile application cloud offloading [29]. For example, the execution time of a process (partition, independent module or an entire application) depends on the mobile specifications and the cloud specifications in addition to the input data size, which complicates the profiling process. The offloading time depends on the communication overhead on the mobile and the cloud (request preparation, communication, and result integration) in addition to the characteristics of the communication channel. Both the computing energy consumption and the offloading energy consumption depend on the mobile specifications, and the latter depends on the communication link characteristics too. Unfortunately, mobile vendors do not provide accurate energy consumption information regarding computation and communication. Energy consumption computation using, for example, external hardware is only valid for the monitored mobile model. Additionally, the offloading objective depends on the process profiling results and its QoS or QoE requirements and hence different offloading techniques with different objectives are needed to suit various processes. Those techniques need to be context aware. For example, it is not unusual that the communication channel quality is unstable, which affects profiling results. The mobile specification can also change, for example when the user switches to a different mobile than the one used for profiling. In other words, static offloading decisions can easily fail in many scenarios [3]. Unfortunately, dynamic partitioning and profiling at runtime can consume considerable computational power and needs to be done in a timely fashion. In general, automated selection of an appropriate offloading technique for each process is a challenge. In addition, different mobile phones have different specifications, and hence it is inevitable that the mobile platform may differ from the cloud platform. CAMO exploits the nature of typical context-aware mobile applications, which involve independent tasks to address some of those challenges as explained in the next section.

G. Discussion

To sum up, an inherent property of typical context-aware mobile applications is that they are essentially partitioned into local partitions that are responsible for checking the mobile context or for effecting mobile adaptations and off-loadable tasks that may run either locally or remotely on the cloud according to their objectives and context. This implies that complex application partitioning techniques (requiring annotations and/or complex representations) are not required for such applications. Additionally, programmers can profile each off-loadable task in isolation on the mobile and the cloud to determine conditions under which it *would* be offloaded rather than merely whether it *should* be offloaded. There is also no need for application migration and replay techniques since those tasks are, typically initiated from scratch at runtime and so do not call for suspending and resuming or replaying. Nevertheless, offloading should be flexible enough to suit various tasks with different objectives in different contexts. In other words, offloading itself needs to be context-aware since as previously noted, it is not always advantageous. Programmers have to plan task offloading with care to avoid unnecessary computations at runtime to save computing power, memory, and energy and avoid delay. CAMO considers this as explained in the following section.

III. CAMO DETAILS

In this section, we explain the details of CAMO showing how it exploits the nature of typical context-aware mobile applications to address some of their offloading challenges.

A. Difference between Mobile and Cloud Platforms

One of the challenges facing context-aware mobile application task offloading is the inevitable possible difference between the mobile specifications and the cloud specifications. In a typical context-aware mobile application, the independency of the tasks increases the chance of finding suitable apps, classes, and libraries for executing them. Accordingly, we accept this difference and do not enforce similarity between the corresponding tasks on the mobile and the cloud as long as the task is adequately executed on both platforms. For example, we can use a mobile app to perform a specific task on the mobile, and use code written in an alternative language for an alternative operating system on the cloud to perform the same task. We provide an example of such a task in Section IV.

B. Resource Consumption Estimation

Another challenge is the estimation of the resource consumption of the tasks when running locally and when running remotely. As previously noted, programmers can exploit the nature of a typical context-aware mobile application, which is composed of independent tasks and use CAMO to profile each task in isolation on the mobile and the cloud. Accordingly, programmers can obtain a rough estimate of its resource consumption such as the execution time, storage space, network usage, and energy consumed when executed remotely and when executed locally under various possible data sizes. Examples of tasks profiling are provided in Section 4.

C. Making Offloading Decisions

To make offloading decisions for each task, we draw a decision tree based on the profiling results. An example decision tree of a text-to-speech conversion task is provided in Figure 1. According to this decision tree, offloading takes place when the battery level is low. Even if remote execution fails, local execution will not take place because the battery does not allow it and merely an error message will be generated. The same applies to the situation in which there are not enough resources (other than energy) for local execution.

On the other hand, even when there are enough power and resources, offloading can still take place to save mobile resources such as the storage space provided that the data size is less than *n* to avoid too much delay. Nevertheless, in the case of an error in remote execution, local execution can take place since there are enough power and resources for this purpose. Finally, in the case of successful remote execution, we check whether the offloading objective corresponding to the offloading criterion is satisfied. In the current example, the objective of considering only data size less than *n* is to avoid intolerable delay. If the objective is unsatisfied due to changes in the mobile environment such as the mobile specifications or the Internet connection between the mobile and the cloud, the criterion can be updated by reducing the value of *n*, for example by a pre-specified percentage. Similarly, if the delay is reduced, *n* may be increased by a pre-specified percentage (provided that the cloud can handle execution with a larger data size).

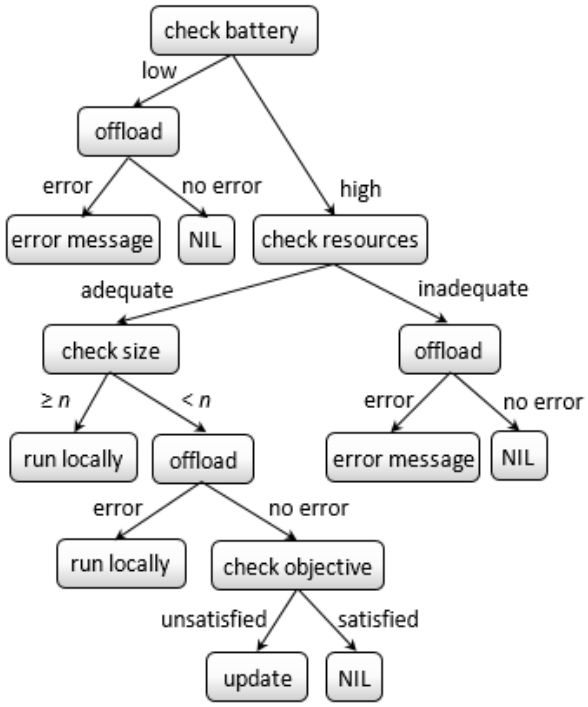

Fig. 1. An example decision tree

D. Task Offloading Plan

One of the challenges of task offloading is that different tasks have various objectives and QoS or QoE requirements. Additionally, each task can have different objectives in different contexts. In this paper, we introduce the concept of task-offloading plans. Figure 2 shows the flowchart of such a plan. As previously noted, a task is triggered when its corresponding context is satisfied. It is worth noting that the application can continuously check the satisfaction of the triggering context of each task as shown in the flowchart or may request notification from CAMO when the context is satisfied.

As shown in the figure, the offloading plan starts by checking whether the programmer enabled offloading and whether there is a connection between the mobile application

and the cloud. Next, the offloading criterion is checked. If it is satisfied, the task can be offloaded. In the case of an error in remote execution, one of two possible actions can take place. Unless the reason behind offloading the task is that local execution was impossible, we can replace remote execution by local execution.

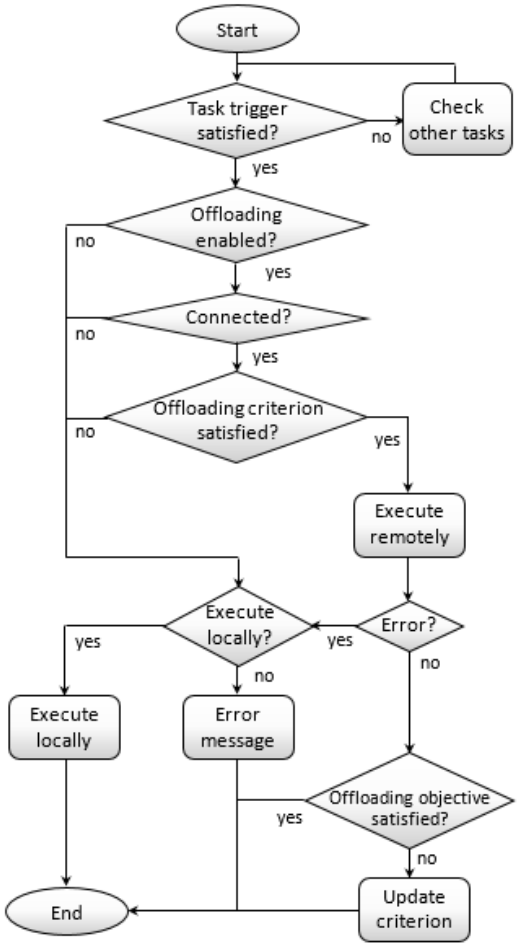

Fig. 2. Flowchart of a task-offloading plan

On the other hand, in the case of successful execution of the task remotely, the objective behind the offloading criterion may be checked to ensure that the criterion is still valid or to update it if required. Given that a typical context-aware mobile application is idle most of the time (when none of the contexts of the different tasks is satisfied), this would have minimal effect on the application.

It is obvious that a task-offloading plan is highly flexible facilitating the development of custom plans that suit various tasks and possibly more than one plan for each task. Table 1 shows the three offloading plans of the text-to-speech task whose decision tree is depicted in Figure 1. In the first two plans, the offloading objective is always true and so nothing is checked. Programmers can use the classes and methods of CAMO to develop such plans as needed. CAMO examines the plans of each task one by one in order to decide whether offloading should take place.

It is worth noting that programmers can replace the task-offloading criterion by the offloading objective (so that CAMO checks the objective directly rather than checking the criterion) as in the case of MobiByte [37]. The side effect is that more time (delay) and power will be wasted as runtime. Choice of whether to check the criterion or the objective depends on the frequency with which the mobile environment (such as the specifications of the mobile and Internet connection) changes.

TABLE. I. THREE OFFLOADING PLANS OF THE TEXT-TO-SPEECH TASK

(a)	***Offloading criterion:*** *battery low* ***On error:*** *error message* ***Offloading objective:*** *true*
(b)	***Offloading criterion:*** *inadequate resources* ***On error:*** *error message* ***Offloading objective:*** *true*
(c)	***Offloading criterion:*** *size < n* ***On error:*** *run locally* ***Offloading objective:*** *delay < t*

E. Discussion

To sum up, we developed CAMO in Java for the Android platform providing classes and methods to allow developing context-aware mobile applications with off-loadable tasks. Programmers can exploit the independency of the tasks of a typical context-aware mobile application and use CAMO to profile each task independently on the mobile and the cloud. They can use the profiling results to develop an offloading decision tree and a corresponding set of offloading plans for each task in different contexts of the mobile and the task. They can then use CAMO to implement custom plans or exploit pre-specified plans, criteria (such as a specific data size) and objectives (such as a specific delay or merely execution regardless of the adequacy of the mobile resources). At runtime, CAMO checks the plans of each task one by one to make informed, context-aware offloading decisions.

It is worth noting that changes in the mobile environment may stem, for example from an unstable Internet connection, difference in the Internet connection specifications from one place to another, or difference in the mobile specifications, when the application runs on a mobile different from the one used for profiling. Programmers can consider such changes in the criteria of the developed plans or by checking each objective and updating the corresponding criterion in case of its violation. Alternatively, in the case of frequent changes, offloading objectives can replace offloading criteria at the expense of additional overhead at runtime.

In CAMO, we exploit profiling results of a given task to reserve cloud resources needed for successfully running it under the largest off-loadable data size. The upper limit on size can be set for example such that delay does not exceed *t* sec. Since an inherent property of cloud computing is its ability to offer customised resources, we should not worry about fluctuation in performance when running the task remotely.

IV. Experiments & Discussion

In this section, we present the details of some experiments based on CAMO. The specifications of the mobile used in the experiments are as follows:

- **Chipset:** Qualcomm MSM8996 Snapdragon 820
- **CPU:** Quad-core (2x2.15 GHz Kryo & 2x1.6 GHz Kryo)
- **GPU:** Adreno 530
- **RAM:** 4GB
- Storage: 64 GB
- **OS:** Android OS, v6.0.1 (Marshmallow)

The Internet connection used in the experiments was Wi-Fi with a bandwidth of 500 KB/s. The cloud platform was Amazon Web Services (AWS) Lambda with the default memory size of 512 MB for executing the remote code and Amazon S3 for storage. We experimented with three tasks of different complexity levels. In those tasks, the execution time, storage space and network usage were the resources of interest.

A. Text-to-Speech Task

The first task is a text-to-speech conversion task. On the mobile, we used Android TextToSpeech class [38] while on the cloud we used the IVONA text-to-speech library [39]. As previously noted, we do not have to use the exact code on both platforms as long as the task is adequately executed on both of them. In order to profile the task, we examined the resource consumption due to converting text that ranged in size, from 50 to 2000 words, to speech in the form of an MP3 file. The results are shown in Figures 3, 4 and 5, respectively.

As shown in the figures, remote execution needs longer time in comparison to local execution since the text entered by the user is first converted into a text file that is uploaded to the cloud where it is converted and stored as an MP3 file. In the case of local execution, on the other hand, the entered text is converted to an MP3 file right away. On the other hand, local execution requires a larger storage space since it stores the MP3 file in the mobile device as opposed to remote execution, which requires local storage of only the text file that is uploaded to the cloud. Concerning the network usage, it is zero for the local execution that does not need any Internet connection at all, but ranges from 1.6 KB (50 words) to 47.3 KB (2000 words) in the case of remote execution. Given that offloading 2000 words, for example, requires 47.3 KB of network usage and that the Internet connection bandwidth is 500 KB/sec, the offloading time is estimated to be 94.6 msec; negligible in comparison to the total remote execution time.

Those profiling results helped in determining the offloading plans depicted in Table 1. Since user QoE does matter in this case, 20 volunteers monitored the delay, and according to their assessment, the largest tolerable delay was 10 seconds corresponding to 1450 words. This agrees with the findings of research studies concerned with usability engineering [40]. This implies that text larger than 1450 words should not be offloaded. It is worth noting that profiling also helped in specifying the storage space that should exist on the cloud (in addition to that needed for the remote code) for the application to work smoothly in case offloading takes place.

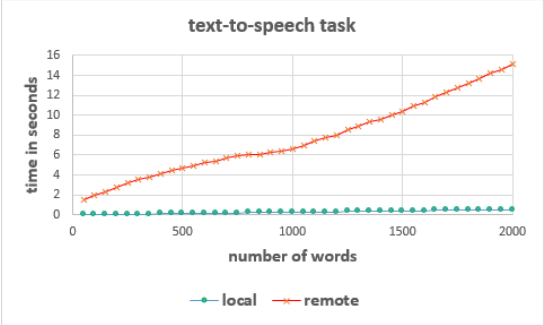

Fig. 3. Execution time (including offloading overhead) of the text-to-speech

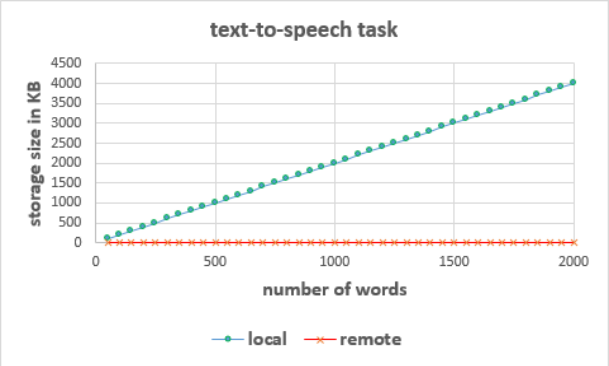

Fig. 4. Storage space of the text-to-speech task

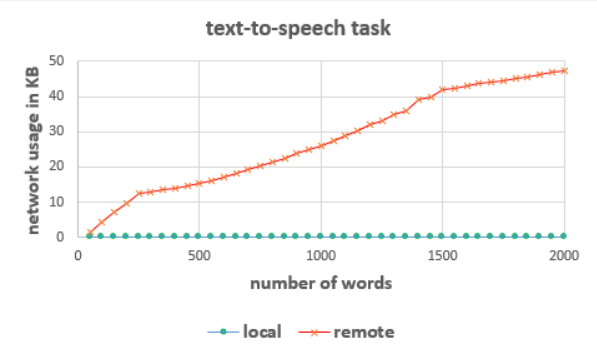

Fig. 5. Network usage of the text-to-speech task

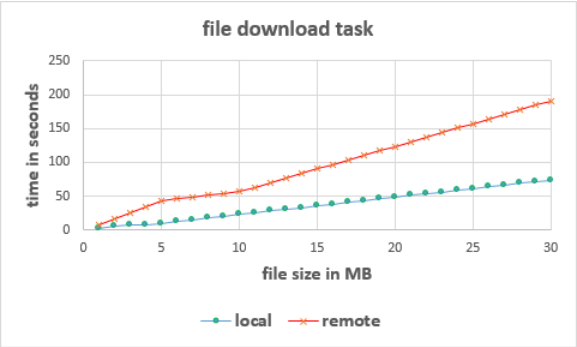

Fig. 6. Execution time (including offloading overhead) of the file download task

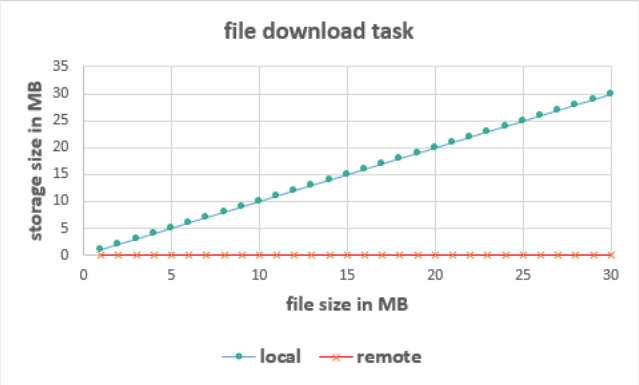

Fig. 7. Storage space of the file download task

Fig. 8. Network usage of the file download task

B. File Download Task

The second task is a file download task. To profile the task, we examined the resource consumption due to downloading MP4 videos that ranged in size from 1 MB to 30 MB. The results are shown in Figures 6, 7 and 8, respectively.

As shown in the figures, remote execution time is much longer than local execution time. The former is roughly about three times the latter. In other words, there is no gain in terms of execution time. It is worth noting that remote code execution uses Amazon Simple Notification Service (SNS) as a trigger to start download. Concerning storage space, a large space is consumed in the case of local execution, but none is required in the case of remote execution, as opposed to the text-to-speech task. The downloaded video is saved on Amazon S3. Concerning network usage, in the case of remote execution, it is no more than 5 KB required for sending the notification. In the case of local execution, it is relatively high.

Those profiling results helped in determining an offloading criterion when the storage space is inadequate. Nevertheless, unless the Internet connection is fast enough, remote execution might be preferable. For example, in the case of a video of size 30 MB and an Internet connection of 500 KB/sec, the time needed for local download is about 60 sec, while the remote execution time is about 190 seconds. In case the Internet connection speed is reduced to 20% of its value, for example, the task would need 300 sec to be executed locally. In this case, remote offloading would be favourable.

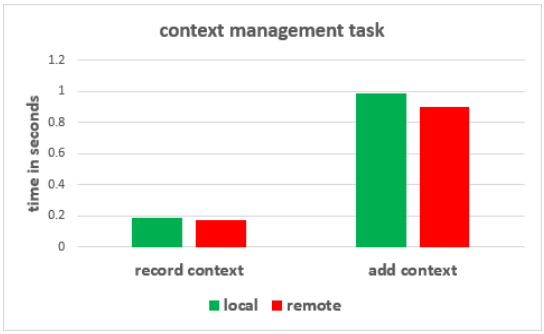

Fig. 9. Execution (including offloading overhead) of the context management task

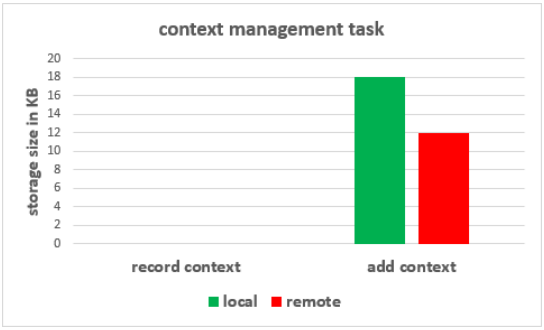

Fig. 10. Storage space of the context management task

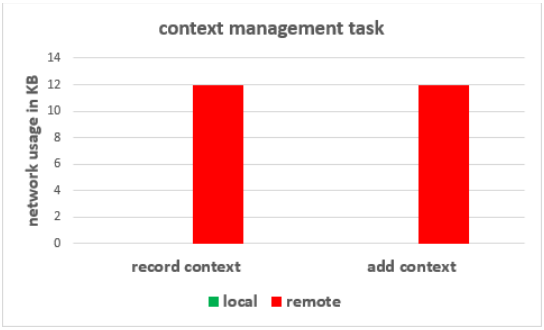

Fig. 11. Network usage of the context management task

C. Context Management Task

In a context-aware mobile application, the context of each task is monitored until it is satisfied triggering the task. In the case of context with monitored sequential values such as time, we can monitor one context value at a time, and as soon as it is satisfied, this event is recorded and the next context value to be monitored is added to a log file. The third task is a simple task that involves recording a satisfied context value and adding the next context value to be monitored to the log file. To profile this task, we executed it both locally and remotely. The results are shown in Figures 9, 10 and 11 respectively.

As shown in the figures, recording context required 0.19 seconds locally and 0.17 seconds remotely. Similarly, adding new context required 0.99 seconds locally and 0.85 seconds remotely. Concerning the storage space, adding context required 18 KB when executed locally and 12 KB when executed remotely due to the difference between the file system on the mobile and the cloud. Recording context, on the

other hand, did not consume significant storage space neither locally nor remotely. Finally, neither adding context nor recording context consumed any network bandwidth during local execution, but required 12 KB of network bandwidth to send a notification to the remote code. It is clear that executing such simple tasks does not benefit much from offloading to the cloud so such tasks better be executed locally.

D. Discussion

In this section, we presented profiling results of three example tasks with variable complexity levels using CAMO. In those tasks, execution time, storage space and network usage were the critical resources of interest. Of course, we could have taken other resources such as power consumption into consideration [41]. We will consider this in future research studies. It is clear that the nature of context-aware mobile applications that involve independent tasks simplifies the profiling process of each task in isolation. Profiling results help in identifying offloading plans and corresponding criteria and objectives that can be implemented using CAMO. In other words, CAMO facilitates developing off-loadable context-aware mobile applications armed with context-awareness.

TABLE. II. EVALUATION RESULTS OF CAMO

Evaluation indicator	**average**
Suitable for context-aware mobile applications	4.66
Useful for saving valuable mobile resources	4.26
May help in developing resource intensive mobile applications	4.80
The concept of offloading plan implies high flexibility to suit various tasks with different objectives in different contexts	4.73
Promising to guide and facilitate the development of efficient context-aware off-loadable mobile applications	4.80
Easy to use	4.66
Would use it in future development	4.60

V. EMPIRICAL EVALUATION

In this section, we present the results of the empirical evaluation of CAMO. We demonstrated the capabilities of CAMO to fifteen Android developers and allowed them to experiment with it and use it for developing off-loadable context-aware mobile applications for two weeks. Afterwards, a questionnaire was provided to assess their satisfaction with it. The developers were asked to provide a response for each item in the questionnaire based on a Likert scale that starts from one (very unacceptable) up to five (very acceptable), and the results are shown in Table 2. As shown in the table, the developers believe that the introduced concept of task-offloading plans deemed to be of high flexibility to suit various tasks and that CAMO is a promising tool for guiding and facilitating the development of efficient off-loadable applications with context-aware offloading decisions. Additionally, it is clear that most of the respondents think it is easy to use, and would readily use it in the future. We estimated the internal consistency and reliability of the questionnaire results using *Cronbach's alpha*. We got an estimated value of α equal to 0.92 signifying an extremely high degree of reliability and recognition of the encouraging questionnaire outcomes.

VI. CONCLUSION

This paper presents a thorough investigation of various techniques, frameworks and development models in the literature to examine their suitability for offloading context-aware mobile applications. Accordingly, the paper proposes the practical context-aware mobile applications offloading development model, CAMO. We developed CAMO in Java for the Android platform providing several classes and methods to help programmers in developing off-loadable applications. Programmers can exploit the independency of the tasks of a typical context-aware mobile application and use CAMO to profile each task in isolation on the mobile and the cloud.

The paper introduces the concept of a task-offloading plan that is highly flexible to suit various tasks with different offloading criteria and objectives. Programmers can use profiling results to develop one or more custom offloading plans for each task in different contexts and use CAMO to implement them. Alternatively, they can use pre-specified plans, criteria and objectives for this purpose. Offloading criteria allow rapid offloading at runtime in case the mobile environment does not change frequently or considerably unlike similar systems that rely merely on the offloading objectives [37]. Finally, CAMO is general enough to allow programmers to use it for any mobile application partitioned into independent modules. Empirical evaluation shows extreme satisfaction of mobile application developers with its promising capabilities.

As future work, we intend to consider other types of resources especially power. Besides, we will explore the idea of incorporating the specifications of the mobile environment including the Internet connection into the offloading plans for increased context awareness rather than merely updating the offloading criteria. Automated generation of offloading plans based on profiling results and given the task objectives in different contexts is currently under development. Such plans can be updated, for example, once before using the application on a new mobile and more often with each considerable change in the Internet connection. In other words, we will continue working on improving CAMO hoping to trigger its wide adoption by mobile application developers to develop efficient applications that can benefit from the cloud.

ACKNOWLEDGMENT

The authors would like to thank both Alaa Alalyani and Malak Alharbi for their help.

REFERENCES

[1] A. Dey, "Understanding and using context," Personal and Ubiquitous Computing, vol. 5, no. 1, pp. 4-7, 2001.

[2] H. Elazhary, A. Althubyani, L. Ahmed, B. Alharbi, N. Alzahrani and R. Almutairi, "Context management for supporting context-aware Android applications development," International Journal of Interactive Mobile Technologies, vol. 11, no. 4, pp. 186-201, 2017.

[3] A. Khan, M. Othman, S. Madani and S. Khan, "A survey of mobile cloud computing application models," IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 393-413, 2014.

[4] H. Elazhary, "Cloud computing for Big Data," MAGNT Research Report, vol. 2, no. 4, pp. 135-144, 2014.

[5] H. Elazhary, "A cloud-based framework for context-aware intelligent mobile user interfaces in healthcare applications," Journal of Medical Imaging and Health Informatics, vol. 5, no. 8, pp. 1680-1687, 2015.

[6] H. Elazhary, "Cloud-based context-aware mobile intelligent tutoring system of technical computer skills," International Journal of Interactive Mobile Technologies, vol. 11, no. 4, pp. 170-185, 2017.

[7] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," Journal of Network and Computer Applications, vol. 48, pp. 99-117, 2015.

[8] T. Verbelen, T. Stevens, F. Turck and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," Future Generation Computer Systems, vol. 29, pp. 451-459, 2013.

[9] B. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," 6th Conference on Computer systems, Salzburg, Austria, pp. 301-314, 2011.

[10] M. Smit, M. Shtern, B. Simmons and M. Litoiu, "Partitioning applications for hybrid and federated clouds," Conference of the Center for Advanced Studies on Collaborative Research, Toronto, Ontario, Canada, pp. 27-41, 2012.

[11] D. Kovachev and R. Klamma, "Framework for computation offloading in mobile cloud computing," International Journal of Interactive Multimedia and Artificial Intelligence, vol. 1, no. 7, pp. 6-15, 2012.

[12] M. Ra, B. Priyantha, A. Kansal and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors," ACM Conference on Ubiquitous Computing, Pittsburgh, PA, USA, pp. 1-10, 2012

[13] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," ACM SIGMETRICS Performance Evaluation Review, vol. 40, no. 4, pp. 23-32, 2013.

[14] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA, pp. 184-194, 2011.

[15] R. Balan, M. Satyanarayanan, S. Park and T. Okoshi, "Tactics-based remote execution for mobile computing," 1st International Conference on Mobile Systems, Applications and Services, San Francisco, CA, USA, pp. 273-286, 2003.

[16] R. Balan, D. Gergle, M. Satyanarayanan and J. Herbsleb, "Simplifying Cyber Foraging for mobile devices," 5th International Conference on Mobile Systems, Applications and Services, San Juan, Puerto Rico, pp. 272-285, 2007.

[17] E. Tilevich and Y. Smaragdakis, "J-Orchestra: Automatic Java application partitioning," 16th European Conference on Object-Oriented Programming, Malaga, Spain, pp 178-204, 2002.

[18] S. Osman, D. Subhraveti, G. Su and J. Nieh, "The design and implementation of Zap: A system for migrating computing environments," 5th Symposium on Operating Systems Design and Implementation, Boston, MA, USA, 2002.

[19] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D. O'Hallaron, A. Wolbach, J. Harkes, A. Perrig, D. Farber, M. Kozuch, C. Helfrich, P. Nath and H. Lagar-Cavilla, "Pervasive personal computing in an Internet suspend/resume system," IEEE Internet Computing, vol. 11, no. 2, pp. 16-25, 2007.

[20] S. Smaldone, B. Gilbert, J. Harkes, L. Iftode and M. Satyanarayanan, "Optimizing storage performance for VM-based mobile computing," ACM Transactions on Computer Systems, vol. 31, no. 2, pp. 5:1-5:25, 2013.

[21] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield, "Live migration of virtual machines," 2nd Symposium on Networked Systems Design & Implementation, Boston, MA, USA, pp. 273-286, 2005.

[22] W. Zhang, K. Lam and C. Wang, "Adaptive live VM migration over a WAN: Modeling and implementation," 7th International Conference on Cloud Computing, Alaska, USA, 2014.

[23] T. Bressoud and F. Schneider, "Hypervisor-based fault-tolerance," 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, USA, pp. 1-11, 1995.

[24] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam and B. Weissman, "ReTrace: Collecting execution trace with virtual machine deterministic replay," 3rd Annual Workshop on Modeling, Benchmarking and Simulation, San Diego, CA, USA, 2007.

[25] J. Tucek, S. Lu, C. Huang, S. Xanthos and Y. Zhou, "Triage: Diagnosing production run failures at the user's site," 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, USA, pp. 131-144, 2007.

[26] J. Flinn and Z. Mao, "Can deterministic replay be an enabling tool for mobile computing?" 12th Workshop on Mobile Computing Systems and Applications, Phoenix, AZ, USA, pp. 84-89, 2011.

[27] A. Surie, H. Lagar-Cavilla, E. de Lara and M. Satyanarayanan, "Low-bandwidth VM migration via opportunistic replay," 9th Workshop on Mobile Computing Systems and Applications, Napa Valley, CA, USA, pp. 74-79, 2008.

[28] S. Hung, C. Shih, J. Shieh, C. Lee and Y. Huang, "Executing mobile applications on the cloud: Framework and issues," Computers and Mathematics with Applications, vol. 63, pp. 573-587, 2012.

[29] A. Khan, M. Othman, F. Xia and A. Khan, "Context-aware mobile cloud computing and its challenges," IEEE Cloud Computing, vol. 2, no. 3, pp. 42-49, 2015.

[30] B. Zhou, A. Dastjerdi, R. Calheiros, S. Srirama and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service," 8th International Conference on Cloud Computing, New York, USA, pp. 869-876, 2015.

[31] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: Making smartphones last longer with code offload," 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, California, USA, pp. 49-62, 2010.

[32] A. Ellouze, M. Gagnaire and A. Haddad, "A mobile application offloading algorithm for mobile cloud computing," 3rd International Conference on Mobile Cloud Computing, Services, and Engineering, San Francisco, CA, USA, pp. 34-40, 2015.

[33] X. Zhang, S. Jeong, A. Kunjithapatham and S. Gibbs, "Towards an elastic application model for augmenting computing capabilities of mobile platforms," 3rd International Conference on Mobile Wireless Middleware, Operating Systems and Applications, Chicago, IL, USA, pp. 161-174, 2010.

[34] R. Kemp, N. Palmer, T. Kielmann and H. Bal, "Cuckoo: A computation offloading framework for smartphones," 2nd International Conference on Mobile Computing, Applications and Services, San Francisco, CA, USA, pp. 59-79, 2010.

[35] V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg and B. Lee, "μCloud: Towards a new paradigm of rich mobile applications," 8th International Conference on Mobile Web Information Systems, Niagara Falls, ON, Canada, 2011.

[36] P. Yuan, Y. Guo and X. Chen, "Uniport: A uniform programming support framework for mobile cloud computing," 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, San Francisco, CA, USA, pp. 71-80, 2015.

[37] A. Khan, M. Othman, A. Khan, S. Abid and S. Madani, "MobiByte: An application development model for mobile cloud computing," Journal of Grid Computing, vol. 13, pp. 605-628, 2015.

[38] TextToSpeech, https://developer.android.com/reference/android/speech/tts/TextToSpeech.html, [Online; accessed: 2017-03-01].

[39] Text-to-Speech, https://www.ivona.com/us/about-us/text-to-speech/, [Online; accessed: 2017-03-01].

[40] J. Nielsen, Usability Engineering. Morgan Kaufmann, 1993.

[41] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," 8th International Conference on Hardware/Software Codesign and System Synthesis, Scottsdale, AZ, USA, pp. 105-114, 2010.