Towards an Offloading Approach that Augments Multi-Persona Performance and Viability

Hanine Tout, Chamseddine Talhi, Nadjia Kara
Department of Software Engineering and
Information Technology
École de Technologie Supérieure
Montreal, Canada
Email: hanine.tout.1@ens.etsmtl.ca
{chamseddine.talhi, nadjia.kara}@etsmtl.ca

Azzam Mourad
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
Email: azzam.mourad@lau.edu.lb

***Abstract*—Mobile virtualization is a key technology that is witnessing widespread adoption to realize multi-persona functionality capable of accommodating work, personal, and mobility needs on a single mobile terminal. Yet, unlike virtualization on servers and desktop machines, mobile virtualization is more challenging due to the limited resources on mobiles platforms in terms of CPU, memory and battery. The evolution of mobile virtualization ranged from heavy to more lightweight techniques capable of running virtual environments on mobile devices with lower overhead. Even though the latest proposed lightweight approaches were able to realize multi-persona, yet none of them is capable of efficiently managing personas performance or ensuring their viability. In parallel, to address the resource limitations of mobile platforms, many researchers have proposed offloading techniques to migrate computation intensive components out of the mobile device to be executed on resourceful mobile cloud computing infrastructure. Motivated by their promising results, we propose in this paper the integration of offloading in the virtual environments on the mobile device toward augmenting personas performance and ensuring their viability. Our experiments show very promising results in this regard.**

I. Introduction

The rapid innovation in electronics over the last few years, and especially in the coming of age of smartphones and tablets, has changed the concept of mobile devices from primitive gadget to full computers that accommodate work, personal and mobility needs. Out of this evolution, the BYOD trend or Bring Your Own Device, has emerged across a variety of industries, as a policy that allows end-users to use personally owned mobile devices for business tasks. Yet, the contrast between business and personal perspectives concerning manageability and security has raised a battle over the adoption of such policy. Dual persona mobile devices were released enabling two phones-in-a-phone, one for private personal use and another for business use, an approach to support BYOD.

Yet, profession and preference have begun to dictate changes in traditional work and personal models pushing mobile devices toward multi-persona functionality capable of consolidating more than two devices in a single device to meet with different users needs [1]. Efficiently managing banking commerce, emails, business and events, preventing social media and untrusted applications from accessing critical information, sharing the device with children for entertainment without ending up with accidental phone calls or unintended in-app purchases, is one of many scenarios in our daily life that drives toward multi-persona [2]. Another good example pushing in the same direction, is in the healthcare field. While working both at their private clinic and at multiple hospitals, doctors are subject to different mobile policies, reflecting each of the different institutions. Managing multiple mobile devices to accommodate with different systems drains their productivity [3]. Whereas, a personal persona, a clinic and hospitals personas, allow doctors to comply with the policies of each, effectively treat their patients while also maintaining their own unburdened personal use of the device [4].

Mobile virtualization is one of the key technologies applied to realize multi-persona. Similar to virtualization on servers and desktop machines, mobile virtualization allows to create multiple virtual environments that live alongside on a single terminal, where in this case, the latter is a mobile device and the environments are called personas. Yet, mobile virtualization is more challenging due to the fact that mobile platforms have limited resources in terms of CPU, memory and battery. Therefore, in the last few years, researchers have proposed lightweight virtualization techniques [2], [5], [6] towards mitigating the virtualization overhead on mobile terminals. Nevertheless, none of these approaches is yet capable of efficiently managing the device performance with respect to the number of personas, and type of applications running in each to realize the aforementioned scenarios. To shed the light on this problem, we compare the device performance while varying the number of personas running diversity of lightweight and heavy applications. Our experiments on Cells [2], which is the first open-source virtualization architecture that enables multiple virtual smartphones and tablets to run simultaneously on the same physical device [7], show drastic increase in the CPU usage, energy consumption as well as in the execution time of the running applications.

On the other hand, in order to address the resource limitations of mobile platforms, many researchers [8]–[11] have proposed offloading techniques to migrate computation-intensive tasks, methods or services out of the mobile device to be executed on resourceful remote servers in a mobile cloud computing infrastructure. The proposed offloading approaches have been able to extend the battery lifetime of the device and offer better performance. Motivated by their promising results, we propose offloading as a strategy to be integrated

978-1-4799-6390-4/15/$31.00 ©2015 IEEE

with mobile virtualization in order to enhance the performance and ensure the viability of multiple personas. The aim of this paper is to demonstrate the feasibility and effectiveness of our proposition by carrying out a study of the offloading's impact on multiple personas running on a mobile platform. Our experimental results show how promising such approach is, in terms of decreasing the CPU usage, energy consumption and the execution time of the applications and ensuring the viability of the running personas.

The contributions of this paper are twofold:

- Revealing the inefficiency of current mobile virtualization approaches in managing the performance and viability of multiple personas running on a single mobile device.
- Proposing the integration of offloading in mobile virtual environments and demonstrating its ability to augment both, performance and viability of multi-persona.

The rest of the paper is organized as follows. Section II presents multi-persona functionality and mobile virtualization technology. In Section III, we highlight the problem caused by running multiple personas on a single mobile device. Section IV illustrates our proposed solution. In Section V, we carry out a study to demonstrate the effectiveness of our proposition and we discuss the relevant experimental results. In Section VI, we review existing relevant approaches. Finally, Section VII concludes the paper and draws our future research directions.

II. Multi-Persona and Mobile Virtualization

Dual persona mobile devices have emerged in the last few years as a technology that supports two-phones-in-a-phone, combining work and personal phones on a single device. Yet the new trends in mobile devices are emerging toward multi-persona, which is the functionality of consolidating multiple mobile devices on a single hardware like multiple-phones-in-a-phone or multiple-tablets-in-a-tablet, rather than just two. Compared to dual persona, it is more complex and challenging due to the extended number of personas running on the mobile platform of limited resources.

To realize multi-persona, a mobile virtualization technique is typically applied. Figure 1 depicts all different techniques that can be used to implement mobile virtualization.

System-Level Virtualization [12] is a virtualization technique that offers the ability to run multiple operating systems on one physical device using an additional software layer called a hypervisor (or microkernel) [6]. This technique is known to have high overhead due to the complete software stack in each virtual environment (i.e., Kernel, Middleware, Apps). Therefore, it is suitable for dual-persona but not multi-persona functionality.

Next, there is User-Level Isolation [13], in which both the kernel and the middleware layers are shared between the virtual instances. Yet this technique does not create virtual environments but rather wraps applications to separate them from each other, which does not realize multi-persona.

The last technique is OS-Level Virtualization, also called Container-based virtualization, which is a virtualization method that shares the kernel layer to run multiple virtual instances on a single operating system [2]. Allocating a minimum set of resources for each persona results in a collection of lightweight personas inside the device making the available OS resources enough for running more than two personas and thus the only technique to realize multi-persona functionality. However by the time, these personas will be vulnerable to performance degradation and would require to be scaled up, which is impossible since they are ported on a single mobile device. Hence, efficiently realizing multi-persona necessitates reinforcing the virtual environments with offloading methods that can minimize the usage of the mobile device resources to augment the performance of the running personas and ensure their viability.

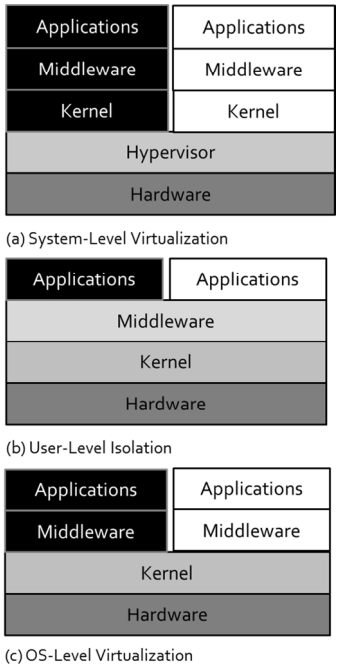

Fig. 1. Virtualization Techniques

III. Problem Statement

The successful deployment of multi-persona strategy in many fields cannot hide its performance and viability dilemmas resulting from the limited resources on mobile devices. The insufficient CPU capabilities, memory size and battery lifetime of such platform will sooner or later decrease the performance of the running personas and even worse, might force them to shut down. For instance, in the healthcare field, doctors use their mobile devices to access patients records and collaborate with other physicians. Yet, unlike other professionals, doctors work at multiple hospitals besides their private clinic, where they are subject to different mobile policies, reflecting each of the different institutions. Therefore, looking for a solution that allows for a good fit amongst their multiple places of practice and their patients quality of care, multi-persona mobile device is the winning ensemble [4]. Carrying a single device with persona for each of their practices, allow them to comply with different policies, while effectively treating their patients. Nevertheless, to meet with the demand of their profession, doctors need to access patients records and deliver care no matter

their place for the time being, which necessitates running many personas simultaneously on the mobile device. Subsequently a serious performance degradation and viability problem are likely to arise on this resource constrained platform.

To shed the light on these issues, we compare the device performance while varying the number of personas running diversity of heavy and lightweight applications. To create personas, we use Cells [2] since it is the first open-source virtualization architecture that enables multiple virtual smartphones and tablets to run simultaneously on the same physical device. We set up the environment on an Asus Nexus 7 tablet as Cells open source project has been ported for this device only. The tablet runs Android operating system, has quad-core processor and 1 GB of RAM. In each persona, we consider two benchmark applications:

a) NQueens Puzzle: This application implements the algorithm to find all possible solutions of the typical NQueens problem, and return the number of solutions found. We consider N=13 since with such value the problem becomes computationally intensive.

b) Virus Scanning: The virus scanner scans the contents of some files on the phone against a library of 1000 virus signatures, one file at a time. To implement lightweight application, we fix the size of the files to 100 KB.

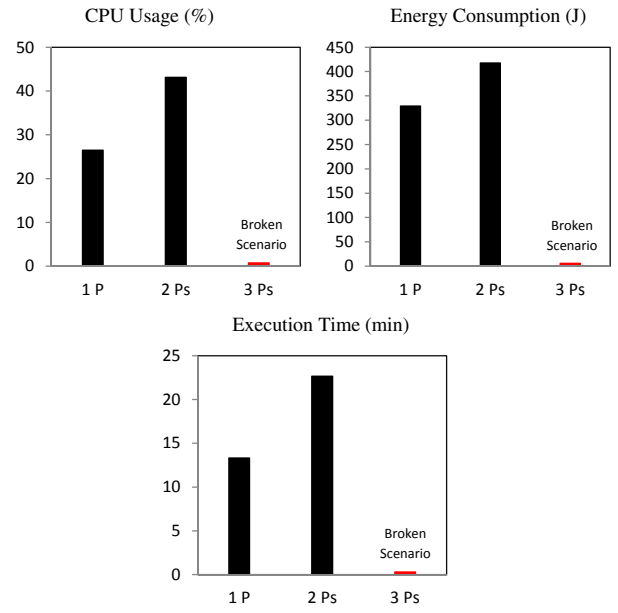

Fig. 2. Multi-Persona Performance

For the sake of these experiments, we implemented a profiler that uses linux-based commands for system calls to collect information about the CPU usage per application. As for energy consumption, we used PowerTutor [14], which accounts, in the following experiments, for power consumed by the CPU and device screen. Finally, we implemented the adequate method to calculate the execution time of these applications.

Figure 2 shows clearly a drastic increase in the CPU usage that was originally 26 % with 1 Persona (1P) but reached 43 % with 2 Personas (2Ps). Also the energy consumption has significantly increased from 329 J to 418 J. As for the execution time, it took 13 minutes to execute both apps in one persona, yet up to 22.6 minutes with two personas. Another interesting observation is in the third scenario (3Ps) where it was impossible to run the same apps in three personas as the personas kept shutting down. These results reveal the inefficiency of the current mobile virtualization approaches in managing the performance and viability of multiple personas as all applications in all personas run locally on a single resource constrained mobile device. In the light of this serious problem, it is indispensable to find a strategy that is able to efficiently manage the running personas with respect to their number and type of applications running in each.

IV. Offloading to Augment Multi-Persona Performance

In this Section, we examine first the offloading techniques that have been introduced in order to address the resource limitations on mobile devices. Then, we present our proposed architecture.

A. Offloading

In mobile cloud computing, offloading is a feature that offers the ability to migrate the execution of applications out of mobile devices to resourceful remote servers. Several offloading techniques have been proposed recently, aiming to improve the battery lifetime of mobile devices and increase the performance of the applications. Motivated by their promising results, we propose in this paper the integration of offloading with mobile virtualization to address the performance and viability issues of multi-persona. The first type of offloading techniques are based on applications offloading, which apply the Pause/Resume model of android [15]. Using the OnPause() method, the application stops running on the mobile device and its state is saved and communicated with a remote server. Then using OnResume(), the application get resumed and continue its execution remotely. Finally the result is communicated back with the physical device. More fine grained techniques [8]–[11] are based either on methods or services migration. In these approaches, the application is partitioned into methods or services and based on certain metrics, particular partitions are considered for offloading. In this paper we use this type of techniques since we aim to offer fine grained solution for multi-persona dilemmas in future work.

On the other hand, offloading can be done either in a static or in a dynamic environment (e.g., takes into consideration changes in the network availability, type and bandwidths). In the latter, applications, methods or services are monitored to formulate an optimization problem, whose solution dictates the offloading decision. Yet, the objective of this work is not to propose a new offloading technique, which we keep for future work, but rather to study the impact of offloading on multi-persona. Therefore, in the following experiments we assume available and stable network, which is a reasonable assumption for the targeted environments like health care institutions.

B. Architecture

Figure 3 depicts our proposed architecture for such study. It consists of offloading the execution of the applications in

multiple personas (P1, P2 and P3 in the figure) to remote server reducing the resource usage on the physical mobile device and hence augmenting the personas performance and viability. In our proposition, mobile applications are assumed to be implemented using the activity/service model in android, where the logic code of the computation-intensive methods is implemented as services through an interface defined by developers using interface definition language (AIDL), and the user interface as activities. This is a fair assumption since it is supported and encouraged by the android platform architecture [16], apart from facilitating the offloading task as services and activities are already isolated. In case the applications do not meet with this requirement, an interface can be easily extracted from the original code as demonstrated in [11].

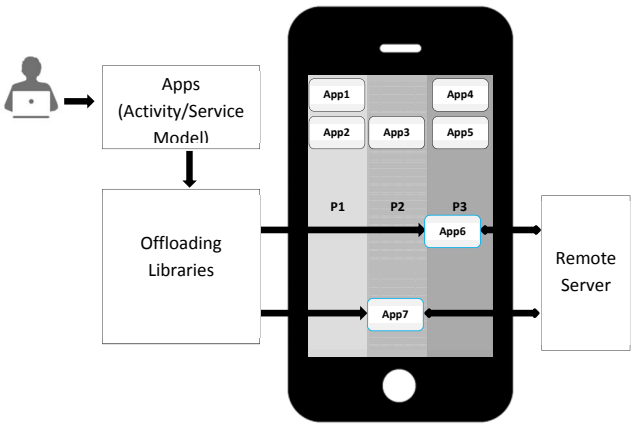

Fig. 3. Proposed Architecture

To make the applications offloadable (App6 and App7 in Figure 3), we use the offloading libraries provided by [11] since it applies the same design pattern (i.e., activity/service). Once invoked, the offloadable service(s) of the application are sent to a pre-configured server for remote execution. To handle the remote invocation, the .class files of the remote implementation will be automatically packaged in a jar and transmitted to the server. Seing that the class files are typically of just few kilobytes, this doesn't cause overhead over the network. In addition, if the requested service(s) are already installed on the server, only the needed parameters are to be transmitted. After being executed on the server, the result is communicated back with the relevant persona(s) on the physical device.

V. Experiments

In the sequel, we discuss our experimental results that demonstrate the efficiency of offloading in multi-persona based on the proposed architecture.

A. Testbed Setup

In the following experiments, we study the impact of integrating offloading in the virtual environments, on multi-persona. We used the same virtualization environment and the same measurement tools of Section III for fair comparison. We also used the same type of applications described in Section III, and we compared the performance of local and remote execution in terms of CPU usage, energy consumption, including energy consumed over the network in the relevant cases, and execution time. In these experiments, we varied both the number of personas and the execution scenarios. As for the network, it is characterized by IEEE 802.11 Standard class n bandwidth and 16348 ms of latency.

B. Results and Analysis

Figure 4 shows the improvement in the personas performance whenever the applications are offloaded. In the first case where only one persona is running on the device (1P), we compared the results among three different scenarios. (1) LL, where both apps (i.e., NQueens solver and Virus Scanning) are running locally in this persona, (2) RL, in which only the heavy application (i.e., NQueens solver) is offloaded, and (3) RR, where both apps are offloaded. As illustrated in the figure, the CPU usage decreased from 27 % to 24 % and reached 2 % running these scenarios respectively. The figure also shows an improvement in the energy consumption that decreased from 330 J to 17 J and to 2.5 J respectively. Also the execution time analysis shows a significant enhancement, where the persona took up to 13 minutes to run both apps while only 5 min in the second scenario and 4min in the third one.

In the second case, two personas are running on the device (2Ps). In the first scenario LL, both apps (i.e., NQueens solver and Virus Scanning) in both personas are running locally on the physical device. In the second scenario RL, one of the personas is running these apps locally while the other is offloading their execution. Finally, in the third scenario, both personas are offloading the execution of both apps to the remote server. The analysis of the CPU usage shows a drastic decrease from 43 % to 34 % and 18 % accordingly. For energy, the consumption decreased from 417 J to 171 J when one of the personas is offloading its apps execution and even to 2.9 J when both personas apply offloading. Same significant improvement in the execution time that got reduced from 22 min to 9.5 min in the second scenario and 4.8 min in the third scenario.

Finally, the last case we took into consideration is the one where three personas are running on the mobile device (3Ps). In this case we tested four different scenarios. (1) LLL, where all personas are running their apps locally, (2) RLL, in which one persona is offloading the applications execution, (3) RRL, where two personas are using offloading and (4) RRR for offloading in all personas. As depicted in Figure 4, it was impossible to realize the first two scenarios where the complexity of the execution has increased with the extended number of personas. It also worth to mention that the running personas kept shutting down. While comparing the last two scenarios, the CPU usage decreased from 36 % to 18 %, the energy consumption from 182 J to 3.7 J. and the execution time from 9.7 min to 5 min.

Our experimental results show clearly that integrating offloading in the virtual environments is really promising toward augmenting the performance and ensuring the viability of the mobile device running multiple personas.

VI. Related Work

In this section, we review first the lightweight mobile virtualization approaches then those relevant to offloading.

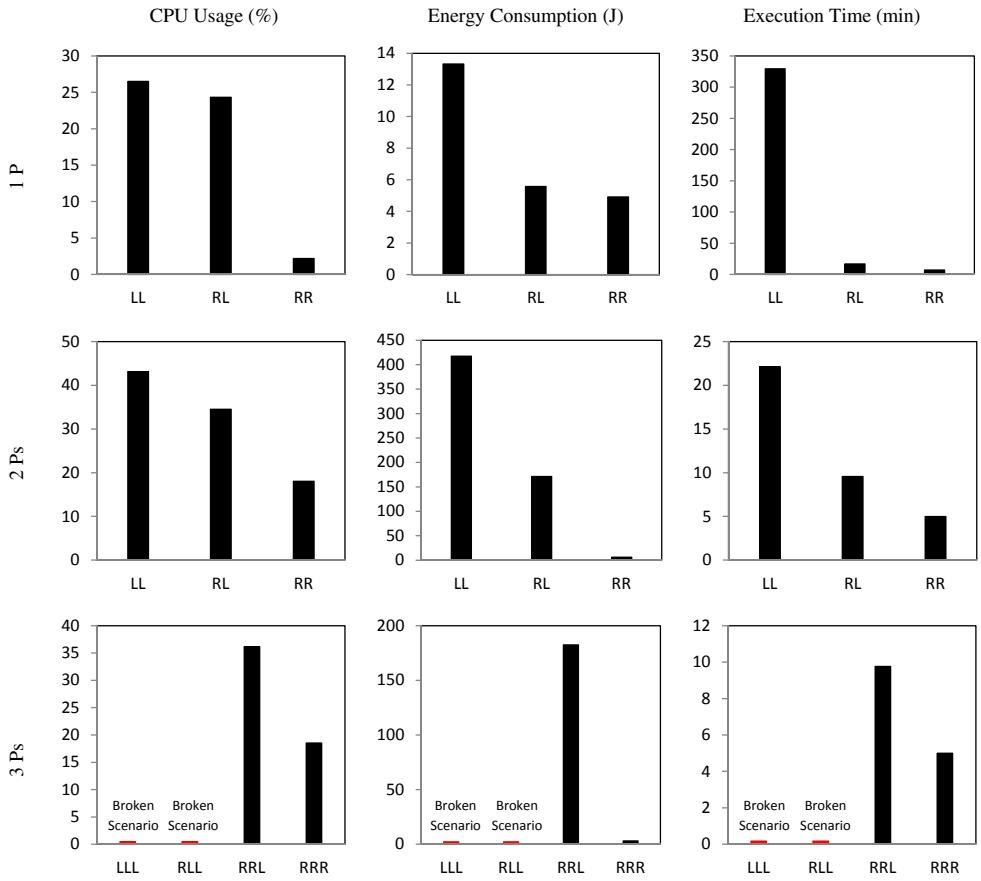

Fig. 4. Offloading Impact on Multi-Persona Performance. For one persona (1P), the experiments are done in three scenarios: LL where both apps are running locally, RL when NQueens app is offloaded, and RR in which both apps are offloaded. For two personas (2Ps): LL means apps are running locally in both personas, RL is when one of the personas is offloading its apps, and RR where both personas are offloading their apps. For three personas (3Ps), the experiments are done in four scenarios: LLL where the three personas are running their apps locally on the mobile device terminal, RLL where one of them is offloading its apps, RRL in which two personas are offloading their apps while the third one has local execution, and RRR where all personas are offloading their apps.

A. *Virtualization Approaches*

Several approaches have been proposed recently in order to apply lightweight virtualization techniques on mobile devices.

TrustDroid [5] is a security framework for lightweight domain isolation on Android, that aims to mitigate unauthorized data access and communication among applications of different trust levels. The architecture exploits coloring of separate and distinguishable components. It colors applications and user data (stored in shared databases) based on a certification that need to be integrated into Android. Based on the applications colors, TrustDroid organizes applications along with their data into logical domains. For lightweight approach, TrustDroid shares both the kernel and the middleware layers among all virtual environments.

In [6], the authors present a lightweight isolation mechanism for Android with access control policies, to separate one or more Android userland instances from a trustworthy environment. The proposed architecture is based on OS-level virtualization, which provides userspace containers to isolate and control the resources of single applications or groups of applications running on top of one kernel.

Another approach that applies the same type of virtualization, which shares the kernel layer among the virtual environments, is Cells [2]. This approach enables multiple virtual phones (VPs) to run simultaneously on the same smartphone hardware. It uses device namespaces to multiplex access among VPs to kernel interfaces and hardware resources such that VPs can run side-by-side in virtual OS sandboxes.

Despite their lightweight techniques, current mobile virtualization approaches are not capable of efficiently managing the performance of the device when running multiple virtual environments (personas). In these approaches, applications in each persona will run locally on the physical mobile device, which causes a serious performance and viability dilemmas on such resource constrained platform as demonstrated throughout this paper.

B. *Offloading Approaches*

In parallel, many researchers proposed offloading frameworks to migrate the applications out of the mobile device to be executed on resourceful remote servers aiming to address the resource limitations of mobile devices.

In [8], the authors address the issue of limited battery power on smartphones. They propose a unified elastic computing plat-

form that combines an infrastructure based cloud and an ad hoc virtual cloud. The proposed platform takes into consideration three different execution strategies. A standalone execution, where all the tasks of an application run on the physical smartphone. A cloud execution, in which the applications run on the cloud. Finally a cooperative execution, where tasks are divided between the smartphone and the cloud infrastructures.

MAUI [9] is a offloading framework that aims to reduce the energy consumption of mobile applications. The framework that consists of (1) a Proxy Server responsible of communicating the method state, (2) a Profiler that can monitor the device, program and network conditions, and (3) a Solver that can decide whether to run the method locally or remotely. In the proposed approach, the mobile application programmer annotates methods that should be taken into account for code offloading. MAUI uses its optimization framework to decide which method to send for remote execution based on the information gathered by the profiler.

ThinkAir [10] aims to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server which communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It also parallelizes method execution by invoking multiple VMs to execute in the cloud in a seamless and on-demand manner to achieve greater reduction in execution time and energy consumption.

Cuckoo [11] is another offloading framework that follows a different strategy for offloading computation-intensive tasks. The proposed framework transforms regular Android application into a computation offloading application. As precondition, all compute intensive code should be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the method call on mobile devices get changed continuously, making offloading sometimes beneficial but not always.

All these approaches were able to prove the efficiency of offloading in managing the performance of resource constrained mobile devices. Their promising results triggered us to introduce offloading as a strategy to be integrated with mobile virtualization to augment the performance and viability of multi-persona.

VII. Conclusion and Future Directions

We demonstrated in this paper the inefficiency of current mobile virtualization approaches in managing the performance and ensuring the viability of multiple personas, while we proved the qualification of offloading in that regard. Our experiments examined promising results of integrating offloading with mobile virtualization, in terms of decreasing the CPU usage, energy consumption and execution time of the running applications in the device personas. They also showed the offloading's capability in running multi-persona scenarios that were not feasible locally even with the existing lightweight virtualizations.

In the light of these results, we can move forward in our work to propose, in the future, a smart offloading strategy that aims to generate the optimal execution configuration for the applications running in each persona and satisfies the personas demands. In our conception, such strategy should gather runtime profiling information to study the behavior of the running applications, and include also a solver, that based on appropriate cost model, different criterion and objectives preferences, can provide optimal partitioning of the applications in all the running personas for local and remote execution.

References

[1] O. Laadan. Multi-persona android. http://events.linuxfoundation.org/sites/events/files/slides/builders-2014-pub.pdf. Accessed: 2014-09-21.

[2] J. Andrus, C. Dall, A. V. T. Hof, O. Laadan, and J. Nieh. Cells: a virtual mobile smartphone architecture. In *Twenty-Third ACM Symposium on Operating Systems Principles*, pages 173–187. ACM, October 2011.

[3] Meru Networks. Byod in healthcare: Improving clinician productivity and patient satisfaction. Accessed: 2014-09-21, May 2013.

[4] O. Eiferman. How to balance security and freedom in medical byod. http://health-information.advanceweb.com/Features/Articles/Physicians-Mobile-Devices.aspx, August 2014. Accessed: 2014-09-21.

[5] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A. R. Sadeghi, and B. Shastry. Practical and lightweight domain isolation on android. In *1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 51–62. ACM, October 2011.

[6] S. Wessel, F. Stumpf, I. Herdt, and C. Eckert. Improving mobile device security with operating system-level virtualization. *Security and Privacy Protection in Information Processing Systems*, pages 148–161, 2013.

[7] J. Andrus, C. Dall, A. V. T. Hof, O. Laadan, and J. Nieh. Cells: Lightweight virtual smartphones. http://systems.cs.columbia.edu/projects/cells/. Accessed: 2014-09-21.

[8] W. Zhang, Y. Wen, J. Wu, and H. Li. Toward a unified elastic computing platform for smartphones with cloud support. *IEEE Network*, 27(5):34–40, 2013.

[9] E. Cuervo, A. Balasubramanian, D. K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, June 2010.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM*, pages 945–953. IEEE, March 2012.

[11] R. Kemp. *Programming Frameworks for Distributed Smartphone Computing*. PhD thesis, VRIJE UNIVERSITEIT, http://dare.ubvu.vu.nl/bitstream/handle/1871/50612/dissertation.pdf?sequence=1, 2014. Accessed: 2014-09-21.

[12] B. Ken, Stephen D. Prashanth B., Viktor G., Perry H., Craig N., Harvey T., and Bruno Z. The vmware mobile virtualization platform: is that a hypervisor in your pocket? *ACM SIGOPS Operating Systems Review*, 44(4):124–135, 2010.

[13] Divide. https://www.divide.com/features/workspace. Accessed: 2014-09-21.

[14] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, October 2010.

[15] S. H. Hung, J. P. Shieh, and C. P. Lee. Virtualizing smartphone applications to the cloud. *Computing and Informatics*, 30(6):1083–1097, 2012.

[16] E. Chen, S. Ogata, and K. Horikawa. Offloading android applications to the cloud without customizing android. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 788–793. IEEE, March 2012.