Mobile Offloading Framework: Solution for Optimizing Mobile Applications Using Cloud Computing

Henryk Krawczyk, Michał Nykiel(✉), and Jerzy Proficz

Faculty of Electronics, Telecommunications and Informatics,
Gdansk University of Technology, Narutowicza 11/12, 80-233 Gdansk, Poland
hkrawk@eti.pg.gda.pl, {mnykiel,jerp}@task.gda.pl

Abstract. Number of mobile devices and applications is growing rapidly in recent years. Capabilities and performance of these devices can be tremendously extended with the integration of cloud computing. However, multiple challenges regarding implementation of these type of mobile applications are known, like differences in architecture, optimization and operating system support. This paper summarizes issues with mobile cloud computing and analyzes existing solutions in this field. A new innovative approach consisting of an application model and a mobile offloading framework are considered and adopted for practical applications.

Keywords: Cloud computing · Mobile · Network · Optimization · Model · Framework · Offloading

1 Introduction

The capabilities of mobile devices such as smartphones, tablets and wearables are increasing at very fast pace. Average processing power of a smartphone increased almost 4 times from 2011 to 2014 [1]. Mobile network connection speeds more than doubled in 2013 – the average mobile network downstream speed was 526 kbps in 2012 and almost 1.4 Mbps in 2013 [2]. It's estimated that by 2018 there will be over 7.4 billion mobile devices with 3G or 4G connection speed and global mobile data traffic will exceed 15 exabytes per month. This is all result of an increased demand from users, expecting continuous access to Internet services, multimedia, social networks, etc.

Arguably the most important issue in mobile devices currently is battery life. While CPU (Central Processing Unit) power, memory size, screen size and number of sensors increased significantly, we're not seeing any noticeable increase in battery capacity. For example the first iPhone, released in 2007, had a 5180 mWh battery and the iPhone 5 s model, released in 2013, have a 5966 mWh battery. The huge increase in processing capability over the 6 years has come with only a 15 % increase in battery capacity

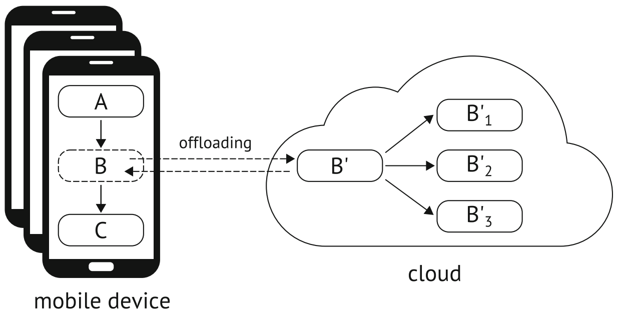

Fig. 1. The concept of offloading a part (component B) of mobile application consisting of three components A, B and C from a mobile device to the cloud

Although processing power of smartphones and tablets are increasing very rapidly, they are still far behind desktop computers or even laptops. Low performance on tasks such as image and video processing, 3D modeling or processing large amounts of numerical data prevents users from replacing their PCs with smaller and cheaper devices.

In recent years these problems have been addressed by integrating cloud computing with mobile devices [3]. Thanks to a new generation of mobile networks the applications are able to transmit large amounts of data to the cloud-hosted services. The concept of using remote servers and resources to extend capabilities of smartphones and tablets enables computationally intensive tasks to run efficiently while minimizing battery usage.

Mobile devices have many features and applications nowadays, from taking pictures, recording videos and playing games to advanced augmented reality software. With the computing power and storage space of the cloud these features may be further extended and new applications are possible. This concept is known in literature as computation offloading and refers to transferring certain processing tasks from devices with hardware limitations to external machines with more computational power [4]. Figure 1 presents the concept of using cloud for offloading of data processing from mobile device. Application component B can be executed in the cloud as parallel tasks B'_1, B'_2 and B'_3. Two advantages of offloading this component will be reduced execution time and less battery usage on mobile device.

However, there are still many unresolved issues with mobile cloud computing and existing solutions don't provide solutions for all of them. A new elastic mobile application model together with an innovative mobile offloading framework that could be adopted for practical applications is presented in this paper. Some examples of applications that could greatly benefit from this approach are multimedia processing [5], mobile gaming [6] and augmented reality [7,8] applications. The cloud offloading could be implemented in many others to improve battery life and performance of mobile devices.

Table 1. Comparison of existing solutions

	Run-time optimization	Application modifications	OS modifications	Other
CloneCloud	Performance	None	Significant	—
Weblets	Composite	Significant	None	Requires using one of supported application patterns
μCloud	None	Significant	None	—
Cloudlet	None	None	Significant	Requires low-latency server, private cloud
eXCloud	None	None	None	—
MAUI	Composite	Small	None	Based on outdated OS
ThinkAir	Energy/ performance	Small	None	Dedicated compiler, offloading only if both objectives are met
Cuckoo	None	Small	None	—

2 Existing Solutions

Multiple frameworks and models designed for integration of cloud and mobile devices exist. They differ in objective, as some of them are designed to maximize performance of an application, others try to optimize energy usage of mobile device by offloading computational-heavy tasks to cloud. There are also several solutions that try to achieve multiple objectives at once. The most popular solutions and differences between them are presented in Table 1.

CloneCloud [9] model is based on cloned version of mobile device's operating system running as a virtual machine on remote server in the cloud. When resource intensive task is started on the mobile device the execution is paused and process state is transferred to the clone. Partitioning component decides in run-time when to offload process to the cloud. Migration is fully automatic and doesn't require any changes in source code by application programmer. However, it requires significant changes in the operating system of mobile device, specifically a new implementation of Dalvik VM [10] in Android. Furthermore, the CloneCloud model optimizes only application performance and doesn't consider energy consumption or transfer cost.

Another cloud integration model assumes that mobile application is built from multiple loosely coupled components – weblets [11]. An application composed from weblets is called elastic application and supports three topology patterns: replication, splitting and aggregation. Elastic application optimization is performed in run-time using cost model, which is based on various parameters. However, authors don't provide any details regarding implementation of the optimization algorithm and example applications use only predefined configurations. Another disadvantage of weblet model is that the application must be built from scratch using provided SDK (Software Development Kit).

In μCloud [12] a mobile application is composed from multiple heterogeneous components. Building an application is reduced to orchestrating existing components to an execution graph. The biggest disadvantage of this model is that optimization must be performed a priori by the developer and no run-time profiling is proposed by the authors. Additionally, new development process requires existing application to be completely rewritten.

Cloudlet [13] concept is based on migrating whole mobile operating system to resource-rich server. When user starts a computationally intensive task virtual machine on mobile device is paused and transferred to remote computer. The main advantage of this model is that it allows any existing mobile application to be migrated without changes in implementation. However, significant modification must be made to mobile operating system. Furthermore, migration of whole virtual machine or even small VM overlay, as authors propose, requires high speed connection and low-latency servers.

Extensible cloud or eXCloud [14] relies on migration of JVM (Java Virtual Machine) stack frames to the cloud. Specialized pre-processor modifies byte code of the application before it is loaded by the JVM. Among the advantages of this solution is that it doesn't require any changes to existing application implementation or mobile operating system. The biggest disadvantage is that eXCloud only performs migration when resources on mobile device are insufficient and doesn't take into account performance or cost of the execution.

MAUI [15] is a framework that is able to optimize execution time and energy use by offloading individual methods to the cloud. Programmer is required to annotate methods allowed for remote execution and the profiler determines in run-time if migration is beneficial. Modifications that must be implemented in mobile application are relatively small, which is one of the biggest advantages of MAUI model. However, MAUI is based on an old .NET Framework version used in outdated Windows Mobile 6.5.

ThinkAir [16] use an Android emulator that allows to execute mobile applications on machines with x86 architecture. Decision whether code should be offloaded to the cloud is based on energy consumption, execution time or both. One of the shortcomings of ThinkAir model is that it doesn't take into account data transfer cost or network speed. Additionally programmer is required to use a dedicated compiler which could be not up-to-date with latest Android version and may not support new features or even fail to compile some applications.

Cuckoo [17] is a very simple offloading framework that was designed to be simple to use for programmers. Application developer provides two implementations of resource-intensive methods: local and remote. Among the advantages of Cuckoo framework is that it doesn't require any changes in operating system. However, offloading decisions in this model are based solely on remote server availability and doesn't consider performance gain, energy usage or migration cost.

3 Proposed Solution

Research and analysis of existing solutions lead to conclusion, that the most common flaws of aforementioned models and frameworks are as follows.

1. Implementation requires use of completely new architecture, patterns or compiler, hence existing applications must be rewritten from scratch.
2. Run-time optimization is not implemented or takes into account only single parameter like performance or battery consumption.
3. The framework requires changes in mobile operating system, making it difficult to use in practical applications without support from manufacturer of the OS.

Considering these issues authors decided that there is a need to design new model of integration between mobile devices and the cloud, together with a programming framework supporting computation offloading. Solution proposed in this paper tries to address aforementioned problems, basing on experiences and results from existing models.

3.1 Design Goals

First of all, proposed model must be elastic and flexible to allow development of different kinds of mobile applications – both simple and complex. Programmer should be able to use various architecture patterns and not be enforced to design application in one way. That implies that any programming interface should be minimal and integrated seamlessly into existing mobile system SDK (Software Development Kit).

Application optimization should be context-aware and should be performed in run-time, as parameters of network, device status or cloud availability could change in any time. Framework should monitor the execution of an application and collect all necessary data like execution time, CPU, memory, network and energy usage, both in mobile device and in the cloud. Optimization method should work on abstract application model and be able to minimize multiple objectives at once.

To maximize potential practical applications of the framework it shouldn't require any changes to the mobile operating system. Any provided tools, compilers or APIs (Application Programming Interface) must be designed as plugins, extensions or libraries, in such a way that future updates of the system wouldn't break existing applications.

3.2 Application Model

To build a feasible mobile application framework the most popular types of applications should be analyzed and used as a foundation to create an universal application model. As a result of an extensive research the following models were identified.

1. Sequential model – control flow in application is sequential, the output from one component is the input of exactly one other component. Simple image processing is an example of this model.
2. Parallel model – same as sequential but the output from one component may be passed to multiple components executing in parallel. Complex image filters, processing of video streams or rendering of 3D scene are good representations of this model.
3. Complex model – components of the application exchange data in arbitrary way, as in any data processing network that could be represented as a DAG (directed acyclic graph). A good example of this model would be an artificial intelligence algorithm.

Because the sequential and parallel models are special cases of the complex model, the later could be used as a general mobile application model. The model consists of multiple components that contains blocks of data processing instructions. It can be represented in form of a directed graph, where nodes are the application components and edges illustrate data flow. Some of these components could be executed either on mobile device or on remote server, other are constrained to the device because they require user input, specific device data or mobile operating system environment. It is also possible that component must be executed in the cloud, for example to exchange data with multiple users.

Mobile application cost model is based on proposed application model. Figure 2 demonstrates a sample application and cost models. If N is a set of components and E is a set of connections between two components the model is defined as follows:

$$M = (N, E, c_{\mathrm{m}}, c_{\mathrm{c}}, c_{\mathrm{t}}, t_{\mathrm{m}}, t_{\mathrm{c}}, t_{\mathrm{t}}) \tag{1}$$

$$E \subseteq \{(n_i, n_j) : n_i \in N \land n_j \in N \land n_i \neq n_j\} \tag{2}$$

$$c_{\mathrm{m}} : N \to \mathbb{R}, \quad c_{\mathrm{c}} : N \to \mathbb{R}, \quad c_{\mathrm{t}} : E \to \mathbb{R}, \tag{3}$$

$$t_{\mathrm{m}} : N \to \mathbb{R}, \quad t_{\mathrm{c}} : N \to \mathbb{R}, \quad t_{\mathrm{t}} : E \to \mathbb{R}. \tag{4}$$

The model contains four functions over set of components:

- execution time on mobile device $t_{\mathrm{m}}(n)$,
- execution time in the cloud $t_{\mathrm{c}}(n)$,
- cost of executing component on mobile device $c_{\mathrm{m}}(n)$,
- cost of executing component in the cloud $c_{\mathrm{c}}(n)$.

For purpose of this paper it is assumed that the cost is equal to energy used during component execution by the device or by remote server. Components that are constrained to the device have infinite cloud cost and cloud execution time, analogically for the cloud constrained components. Two functions over the set of edges (communication) are used in the cost model:

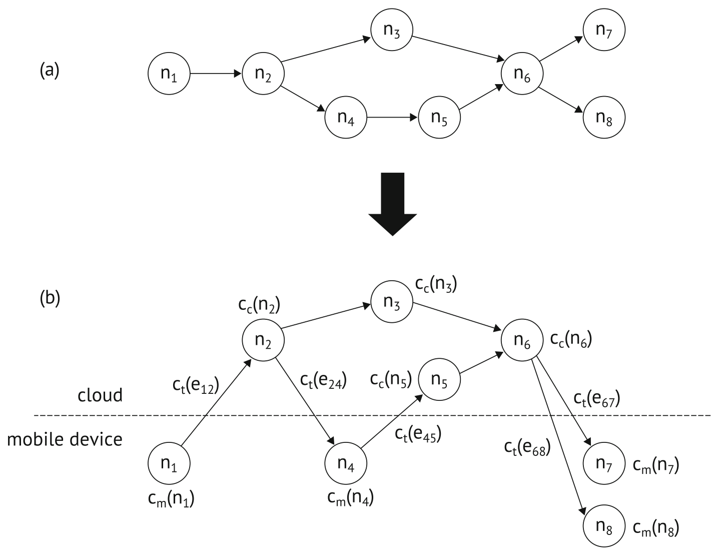

Fig. 2. Example of the application (a) and cost model (b)

- transfer time $t_\mathrm{t}(e)$,
- transfer cost $c_\mathrm{t}(e)$.

Authors assume that time and cost of data transfer between two components executed in the same environment (both on the device or both in the cloud) is negligible and equals zero. If N_m is a subset of components executed on the mobile device and N_c is a subset of components offloaded to the cloud then total cost c of running the application could be calculated:

$$N_\mathrm{c} \subset N, \quad N_\mathrm{m} = N \setminus N_\mathrm{c} \tag{5}$$

$$E_\mathrm{cm} = \{(n_\mathrm{c}, n_\mathrm{m}) : (n_\mathrm{c}, n_\mathrm{m}) \in E \wedge n_\mathrm{c} \in N_\mathrm{c} \wedge n_\mathrm{m} \in N_\mathrm{m}\} \tag{6}$$

$$E_\mathrm{mc} = \{(n_\mathrm{c}, n_\mathrm{m}) : (n_\mathrm{c}, n_\mathrm{m}) \in E \wedge n_\mathrm{c} \in N_\mathrm{c} \wedge n_\mathrm{m} \in N_\mathrm{m}\} \tag{7}$$

$$E_\mathrm{t} = E_\mathrm{cm} \cup E_\mathrm{mc} \tag{8}$$

$$c = \alpha \sum_{n \in N_\mathrm{c}} c_\mathrm{c}(n) + \beta \sum_{n \in N_\mathrm{m}} c_\mathrm{m}(n) + \gamma \sum_{e \in E_\mathrm{t}} c_\mathrm{t}(e). \tag{9}$$

Introducing the α, β and γ coefficients to the equation allows to adjust the cost model to different objectives. Table 2 presents different optimization objectives that can be achieved by assigning different values to the coefficients.

3.3 Optimization Problem

The goal is to find a subset of components to be executed in the cloud that minimizes the sum of total execution cost and transfer cost, while simultaneously preserving total execution time t below a certain threshold t_max:

Table 2. Example of optimization objectives

	α	β	γ
Monetary cost	$\langle 0,1\rangle$	$\langle 0,1\rangle$	$\langle 0,1\rangle$
Mobile device battery usage	1	0	0
Data transfer	0	0	1
Total energy use	$\langle 0,1\rangle$	$\langle 0,1\rangle$	0

$$\min_{N_\mathrm{c}\in\mathbb{P}(N)} \alpha \sum_{n\in N_\mathrm{c}} c_\mathrm{c}(n) + \beta \sum_{n\in N_\mathrm{m}} c_\mathrm{m}(n) + \gamma \sum_{e\in E_\mathrm{t}} c_\mathrm{t}(e). \quad \text{subject to}\, t \leqslant t_{\max} \tag{10}$$

Execution time t is equal to the length of the critical path in DAG representing the application model with component execution times and transfer times used as weights.

The optimization algorithm should compute optimal distribution of components between the mobile device and the cloud, given the cost model as input. Minimization of the cost without constraints could be solved in polynomial time, because the optimization problem could be reduced to minimum cut problem [18]. However, when the execution time constraint is considered the problem becomes NP-hard in general. In special cases of sequential and parallel application models the problem could still be solved in polynomial time. For general approach a genetic algorithm has proved to provide good results for sparse graphs, i.e. application models where the number of edges is significantly lower than square number of components.

4 Mobile Offloading Framework

This paper introduces Mobile Offloading Framework (or MOFF) as an innovative contribution to the field of mobile cloud computing. The idea of MOFF is to provide lightweight middleware both for the cloud and mobile devices that allows dynamic optimization of mobile application cost by offloading parts of data processing to the cloud. Initial implementation of the framework is based on Android operating system and allows of development of mobile applications using both in Java and C++ language. MOFF supports Android version 4.0 and newer which currently covers over 75 % of the mobile market [19].

The idea of MOFF Framework is to enable each component to be executed on mobile device or in the cloud. Application developer is given a programming library that supports development of components in Java or C++ programming language. Developer can prepare one general implementation or different implementations of the same component for mobile and cloud execution to take advantage of resource-rich remote machines. Compiled components are embedded in mobile application package (APK) together with Mobile Service needed for run-time monitoring and communication with the cloud. All components are also deployed to the Cloud Service which supports execution in the cloud.

After implementation of all required components the application developer must orchestrate them and provide the application model which is defined as a JSON document. For example, a definition of the application model from Fig. 2 would look like this:

```
{
    components: [{
        "id": "n1",
        "output": [
            "n2"
        ]
    },{
        "id": "n2",
        "output": [
            "n3", "n4"
        ]
    },
    /* ...definitions of components n3, n4, n5... */
    {
        "id": "n6",
        "output": [
            "n7", "n8"
        ]
    },{
        "id": "n7"
    },{
        "id": "n8"
    }]
}
```

Typical sequence of events in mobile application execution using Mobile Offloading Framework is shown in Fig. 3. When users starts the mobile application (1) Mobile Service gathers information about the device and current execution context (like network connection type, battery level, etc.) via various Android APIs and Linux kernel modules. This data is sent together with application model definition to the Cloud Service (2).

In order to perform accurate optimization the cost model must be constructed on top of the application model. The Cloud Service queries the database for historic data regarding execution cost and time on mobile device and in the cloud for each component (3). The Cloud Service tries to use data for the specific execution context or average from similar contexts in case that the specific one does not exist in the database yet. When the cost model is constructed (i.e. functions of cost and time have values for each component and communication between them) optimization of application model can be performed by genetic algorithm. The result of optimization is a partition of the components set into two subsets: components that should be executed on mobile device and components that should be executed in the cloud.

Results of the optimization are sent back to the mobile device (4) and the application executes components that are assigned to it (5). The Mobile Service

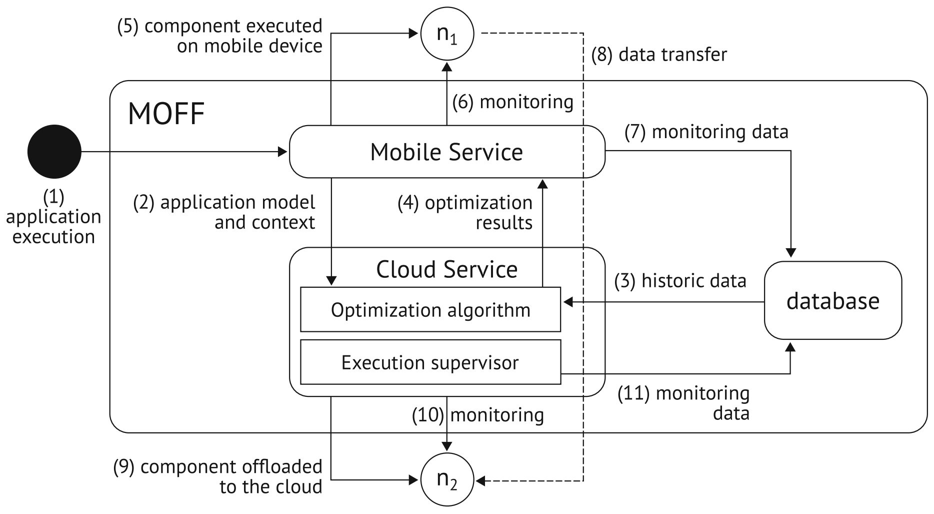

Fig. 3. Sequence of events in application execution with Mobile Offloading Framework. Components n_1 and n_2 are a part of mobile application presented in Fig. 2

must constantly monitor execution time and cost of every component (6). It is important to notice that cost and time functions could be different for every device and depend on current execution context, i.e. network type, battery level, device load, etc. Monitoring data is stored in centralized database (7), therefore historical data from previous executions and other devices could be used as a good basis for future optimizations.

When application tries to execute component that should be offloaded to the cloud Mobile Service communicate with Cloud Service and transfers input data over the network (8). The Cloud Service launches required component on remote server (9), supervises the execution (10) and stores monitoring data in database (11). When component processing is finished the output data is transferred back to the mobile device. The offloading process is transparent for the application.

MOFF Framework introduces a few tweaks to further improve performance of the offloading process. First obvious improvement is to execute two or more consecutive cloud components in one batch rather than sending data to the mobile device after each one. Secondly, the components that are executed in the cloud are started immediately after the optimization is performed and are waiting for input data from the mobile device. This approach helps to decrease delay between sending the data and receiving results from the component because the system process is already initialized.

The Mobile Service includes also some basic fail-over mechanisms. When network connection is unavailable and optimization could not be performed all components are executed on the device if possible. If connection is lost during application execution the Mobile Service takes care of restarting the component locally.

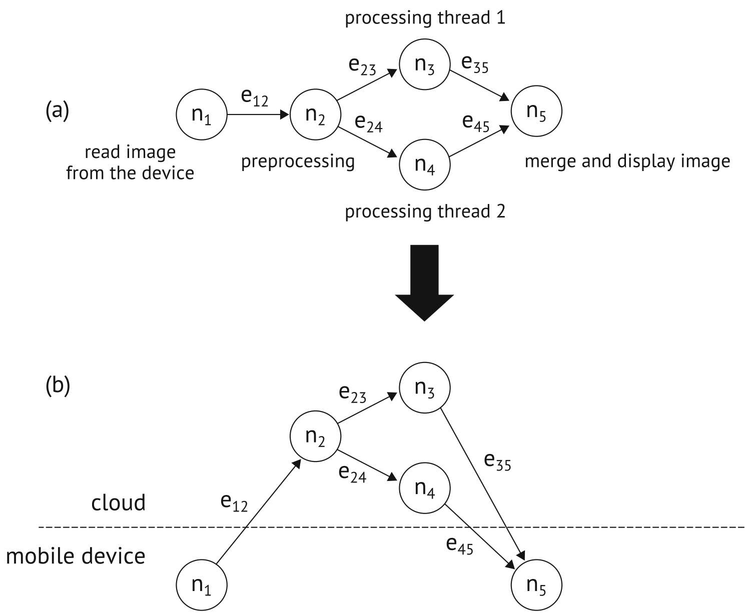

Fig. 4. The application model (a) and optimal component distribution (b) of the image processing application

Table 3. The values of cost [mWh] and time [s] functions measured by MOFF

	c_m	c_c	t_m	t_c		c_t	t_t
n_1	0.04	∞	0.09	∞	e_{12}	0.12	4.06
n_2	0.73	0	1.47	0.31	e_{23}	0.06	4.21
n_3	4.31	0	8.72	1.86	e_{24}	0.06	4.21
n_4	4.31	0	8.72	1.86	e_{35}	0.05	3.22
n_5	0.07	∞	0.08	∞	e_{45}	0.05	3.21

Table 4. Test results of image processing application with and without optimization

	single image		20 images	
	cost [mWh]	time [s]	cost [mWh]	time [s]
not optimized	9.46	10.36	189.21	207.22
optimized	0.33	9.62	6.67	192.41

5 Case Study

MOFF was used to implement sample mobile application for processing images. It consisted of five components: reading an image from the device, preprocessing, applying filters using two parallel threads and displaying processed image.

Two implementations of preprocessing and filtering components were prepared: mobile implementation using OpenCV4Android [20] and cloud implementation using KASKADA services [21]. KASKADA is a distributed platform for processing multimedia streams realized as a part of NIWA Center of Excellence [22]. The framework was used to optimize application using the battery saving objective. The application model and optimal solution is presented in Fig. 4. Table 3 contains values of the cost and time functions measured by the framework. Note that components n_1 and n_5 are restricted to the device. Test results of processing 20 photos on HTC One X device connected to the internet via 3G connection with and without optimization are presented in Table 4. Offloading processing to the cloud reduced battery drain from 189.21 mWh (2.8 % of total battery capacity) to 6.67 mWh (0.1 %) and reduced total processing time by 15 s.

6 Summary

The solution presented in this paper addresses three most common issues with existing models: flexibility of run-time optimization function, programming effort and mobile operating system support. Authors believe that Mobile Offloading Framework has practical use in existing and future mobile applications.

Regarding future research the framework should be more extensively tested by developing more advanced mobile applications. Additionally, more complex execution monitoring could be implemented to improve optimization results in more complicated applications, perhaps by including static code analysis. Support for different mobile operating systems is also worth considering.

In the near future Mobile Offloading Framework will be deployed to the cloud based on Tryton supercomputer that is a part of NIWA Center of Excellence project [22]. Tryton is located in Academic Computer Centre in Gdansk (CI TASK) an is one of the fastest supercomputers in Europe with over 1.2 PFLOPS computing power, 2600 CPUs and 31000 cores. Powered by this infrastructure MOFF could be easily adopted by application developers and scientist to create efficient mobile applications.

Acknowledgments. This work was carried out as a part of the Center of Excellence in Scientific Application Development Infrastructure "NIWA" project, Operational Program Innovative Economy 2007–2013, Priority 2 "Infrastructure area R&D".

References

1. Chitkara, R.: Application processors: driving the next wave of innovation. In: Mobile Technologies Index, PwC (2012)
2. Cisco visual networking index: global mobile data traffic forecast update 2013–2018 (2014)
3. Khan, A.R., Othman, M., Madani, S.A., Khan, S.U.: A survey of mobile cloud computing application models. In: Communications Surveys and Tutorials, vol.16, no. 1, pp. 393–413. IEEE (2014)

4. Kumar, K., Liu, J., Lu, Y.-H., Bhargava, B.: A survey of computation offloading for mobile systems. In: Mobile Networks and Applications, vol. 18, no. 1, pp. 129–140. Springer (2013)
5. Satyanarayanan, M.: Mobile computing: the next decade. In: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS 2010), pp. 1–6. ACM, New York (2010)
6. nVidia GRID game service. http://shield.nvidia.com/grid-game-streaming/
7. Google glass. http://www.google.com/glass/
8. Azuma, R.T.: A survey of augmented reality. In: Presence: Teleoperators and Virtual Environments, vol. 6, no. 4, pp. 355–385 (1997)
9. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer systems, pp. 301–314. ACM (2011)
10. ART and Dalvik. http://source.android.com/devices/tech/dalvik/
11. Zhang, X., Jeong, S., Kunjithapatham, A., Gibbs, S.: Towards an elastic application model for augmenting computing capabilities of mobile platforms. In: Mobile Networks and Applications, vol. 16, no.3, pp. 270–284. Springer, New York (2011)
12. March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., Lee, B.S.: μCloud: towards a new paradigm of rich mobile applications. In: Procedia Computer Science, vol. 5, pp. 618–624. Elsevier (2011)
13. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. In: Pervasive Computing, vol. 8, no. 4, pp. 14–23. IEEE (2009)
14. Ma, R.K.K., Lam, K.T., Wang, C.L.: eXCloud: Transparent runtime support for scaling mobile applications in cloud. In: 2011 International Conference on Cloud and Service Computing, pp. 103–110. IEEE (2011)
15. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: MAUI: making smartphones last longer with code offload. In: Proceeding of the 8th International Conference on Mobile Systems, Applications, and Services, pp. 49–62. ACM (2010)
16. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: Unleashing the power of mobile cloud computing using ThinkAir. In: arXiv preprint arXiv:1105.3232 (2011)
17. Kemp, R., Palmer, N., Kielmann, T., Bal, H.: Cuckoo: a computation offloading framework for smartphones. In: Gris, M., Yang, G. (eds.) MobiCASE 2010. LNICST, vol. 76, pp. 59–79. Springer, Heidelberg (2012)
18. Hao, J.X., Orlin, J.B.: A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms **17**(3), 424–446 (1994). Elsevier
19. Smarthphone OS market share, Q3 (2014). http://www.idc.com/prodserv/smartphone-os-market-share.jsp
20. OpenCV4Android. http://opencv.org/platforms/android.html
21. Krawczyk, H., Proficz, J.: KASKADA - multimedia processing platform architecture. In: Proceedings of the International Conference on Signal Processing and Multimedia Applications, SIGMAP 2010. IEEE (2010)
22. NIWA center of excellence. http://www.niwa.gda.pl