Cloud Computing for Mobile Devices - Reducing Energy Consumption[1] -

Veronika Strokova[2], Sergey Sapegin[3], Andreas Winter[4]

Abstract

Being powered by the batteries that are limited in their capacity is one of the main restrictions of mobile devices. Further enhancement of their characteristics and mobile Internet mounting speed incite the growth of user's demands. Users request the most sophisticated applications to work rapidly and being available all the time. Thus, availability of mobile devices should not be decreased by inefficient energy consumption.

This paper presents the approach which is able to decrease the power consumption on mobile gadgets. The core idea lies in migrating parts of the application's functionality to remote servers in order to reduce energy consumption on the mobile device. Heavy-loaded code blocks are extracted from mobile apps and transferred to server-side applications. It is expected that, if energy spent on client-server communication is less than power needed to execute the task on phone or tablet; battery life time can be extended on the mobile device. Depending on the amount of data, available internet speed, and cloud computing capabilities, systems performance can also be affected.

Experiments are conducted on migrating three Android applications to the cloud. The paper describes the migration approach, shows changes in energy consumption and demonstrates conditions to be met, when doing energy migrations to the cloud, successfully.

1. Introduction and Approach

One of the main restrictions of using mobile devices is their restricted uptime caused by limited battery capacity. Since current battery techniques do not offer means to increase their capacity [8], other strategies to increase the runtime of mobile devices within one battery cycle have to be found. This paper presents an approach to decrease the power consumption on mobile devices by outsourcing parts of an applications' functionality to a remote server in the cloud. Experiments are conducted on three Android applications and show how functionality can be migrated to the cloud, and how this effects systems behavior and energy consumption on the mobile device.

The objectives of this work, which summarizes [12], are to elaborate an approach to migrating parts of mobile applications to the cloud and to define its value from the perspective of saving power on a mobile device. Since further elaborated techniques for optimizing energy consumption in the cloud exist [7], this paper only focusses on the mobile device, mostly aiming at increasing its uptime without further charging the battery.

Applications, suitable for being migrated to the cloud, contain rich functionality in "heavy-loaded" blocks of code, which are responsible for the pivotal functions of the app. But at the same time, these blocks take most of the consumed energy. These may include complicated transformations, calculations, or data processing. The presented approach follows classical software reengineering strategies to improve the applications quality, which here, aim at improving the apps energy consumption. The major reengineering steps include (1) identifying the code to be migrated to the

[1] This work was partially supported by a student exchange grant for eastern European students, funded by DAAD and Rotary Club Oldenburg-Ammerland e.V.

[2]Voronezh State University, 394006 Voronezh, Russia, vesna.adrs@gmail.com, Department of Computing Science.

[3]Voronezh State University, 394006 Voronezh, Russia, svsapegin@mail.ru, Department of Computing Science.

[4]University of Oldenburg, 26111 Oldenburg, Germany, winter@se.uni-oldenburg.de, Software Engineering.

cloud, (2) extracting it from the applications code base, (3) moving it to the cloud, and (4) making it available to the app again by remote access. In such a way, when a user runs an application, a part is still executed on the mobile device, but the energy consuming parts run on a remote server, possibly saving energy on the mobile device. If data transfer to and from the cloud eats less energy than saved by doing the calculations externally, the energy consumption on the device will be reduced, which results in an extend battery live phase.

The core idea of the approach is to delegate parts of functionality from mobile devices to the cloud. To state its effectiveness clearly, the following preconditions have to be ensured:

- The energy needed for data transmission from device to a server and for sending results back to the mobile device must be less than the energy consumed for executing the same functionality at the device;
- The apps' code contains some "heavy-loaded" code blocks which can be identified, extracted, and migrated to the cloud, without influencing the application functionality.

Migrating functionality to the cloud will also affect the performance of the app. That is why we consider the time the operations needed to be executed. Heavy data exchange can slow down the application's speed, whereas high performance calculation in the cloud, e.g. by using cluster computer, can also speed up the application.

2. Experiments

Migrating functionality to the cloud requires finding the "heavy-loaded" blocks of code. It looks sensible to look for:

- code containing complicated calculations which load the CPU significantly and may take a lot of time,
- methods with a few lightweight parameters, but lots of sophisticated functionality inside. As we will see later, you will not benefit from transferring a huge deal of data over the network sometimes,
- existing asynchronous Tasks. All long running operations, on the one hand, should be implemented as asynchronous tasks; on the other hand, it is beneficial to run them remotely. HTTP communication should be done as asynchronous Task in order not to block the app, as the connection is unpredictable.

Three applications which process data locally on mobile devices were investigated: *Finite-ElementCalculator*, *AsciiCamImage* [3] and *AsciiCamVideo*. The presented experiments were accomplished on a Samsung Galaxy Nexus as test device.

The power is measured with Little Eye [10]. It is a desktop performance analysis tool by Little Eye Labs for Android applications. It measures app's power consumption (separating total amount, CPU, WiFi, and display usage), network data transfer and memory usage. Little Eye also shows the key events happening while the application is going on, such as grabbing and releasing the lock, display going on and off, the current app state, etc. After the measurement is stopped, it is possible to return to any time point and explore what was happening. The great feature is an option of building graphic report which is divided into power, data and memory sections.

Each application was run in three modes: on a device, using the cloud with fast internet connection, and using the cloud and slow internet connection.

FiniteElementCalculator performs complex mathematical calculations based on significant but not very large amount of data and uses asynchronous tasks. Due to the absence of platform-specific code and clear modularization of the code, responsible for the required mathematical operations, it was easy to identify and to migrate the calculations to the cloud. The energy consumption was decreased significantly, since only a relatively small amount of data had to be packed into JSON [9] and transmitted over the net.

Table 1 [12] illustrates saving on battery power, CPU resources, and time, while running the application with a server in comparison to the case when only the resources of a mobile device are used (column "Local") Two different ways of connecting mobile device and server were applied (columns "Fast connection" and "Slow connection"). Cases are the amount of user's tasks to be completed. The volume of outgoing and incoming data is deliberately not summarized in the second and third cases to differentiate between calls to the server.

	Local			Fast connection (D 6-12 Mbps/ U 7,7-12)			Slow connection (D 0.9-2.4 Mbps/ U 0,6-0,8)			
	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Data volume (out/in), Mb
FiniteElementCalculator										
1 case										3.2 / 103.34
Average	29.3	31.8	32.3	17.35	3.83	15.3	16.6	2.9	20.3	
Standard deviation	1.26	2.02	1.53	0.7	1.2	7.5	0.15	0.48	4.04	
3 cases										3.2 + 17.92 + 8.16 / 103.68 + 72.24 + 122.12
Average	30.3	36.3	123.3	16.6	3.13	52.3	16.7	2.3	72.3	
Standard deviation	0.41	0.48	4.62	0.39	0.1	3.79	0.25	0.31	9.7	
5 cases										1.63 + 3.2 + 3.2 + 17.92 + 8.16 / 111.79 + 103.34 + 103.68 + 72.24 + 122.12
Average	30	36.4	181.7	16.9	3.2	78.7	16.7	1.87	136.3	
Standard deviation	0.27	0.19	4.62	0.3	0.36	10.4	0.12	0.04	3.79	

Table 1. FiniteElementCalculator power measurement results

Figure 1 and 2 illustrate the energy consumption during one of the use cases. The CPU's and total consumption and time decreased dramatically, the network consumption is not significant in this case.

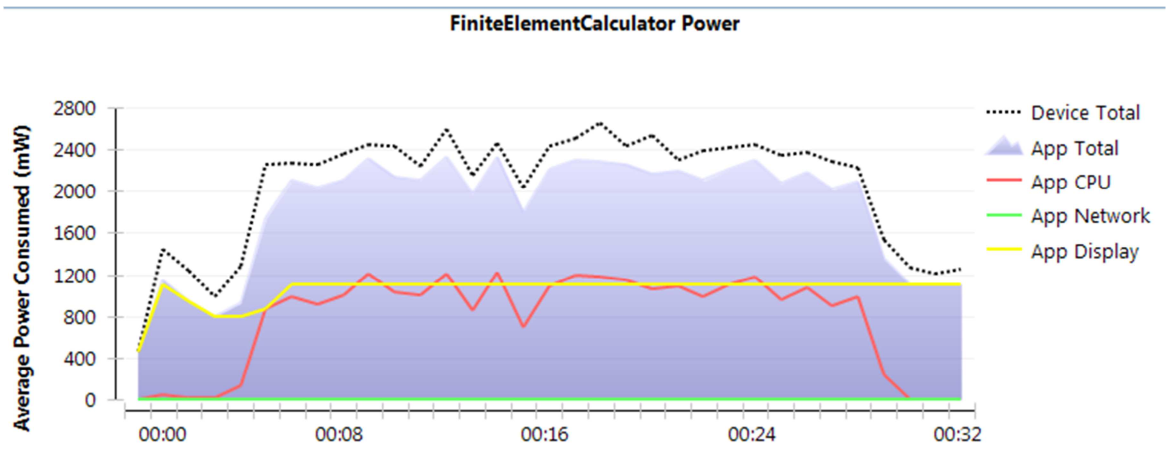


Figure 1. FiniteElementCalculator power consumption while running on a mobile device

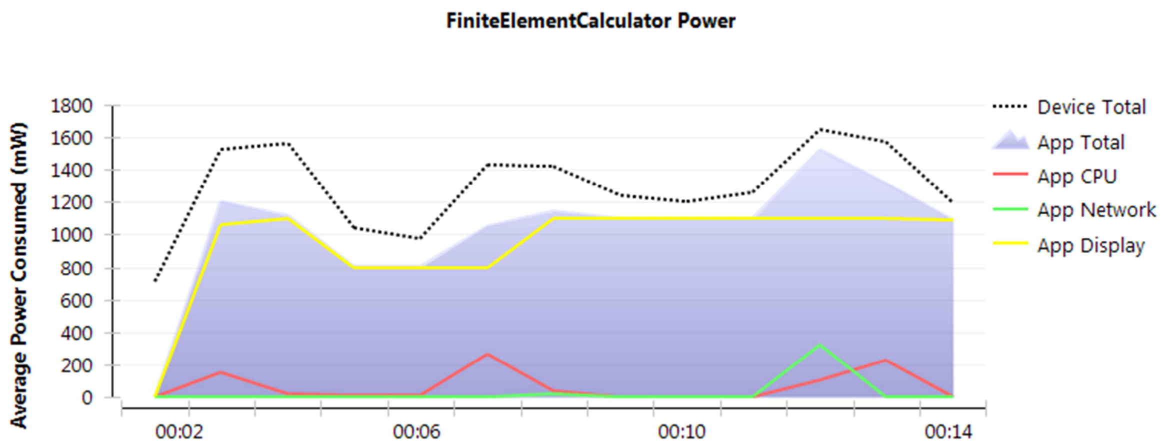


Figure 2. FiniteElementCalculator power consumption while using the cloud

The power consumption decrease is gained by CPU loading relief. Figures 3 and 4 show how the CPU usage changed for both activations.

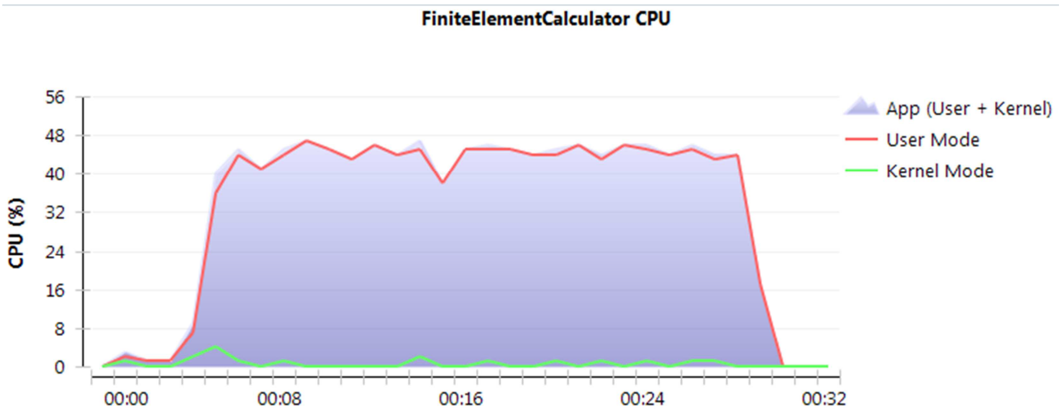


Figure 3. FiniteElementCalculator CPU load while running on a mobile device

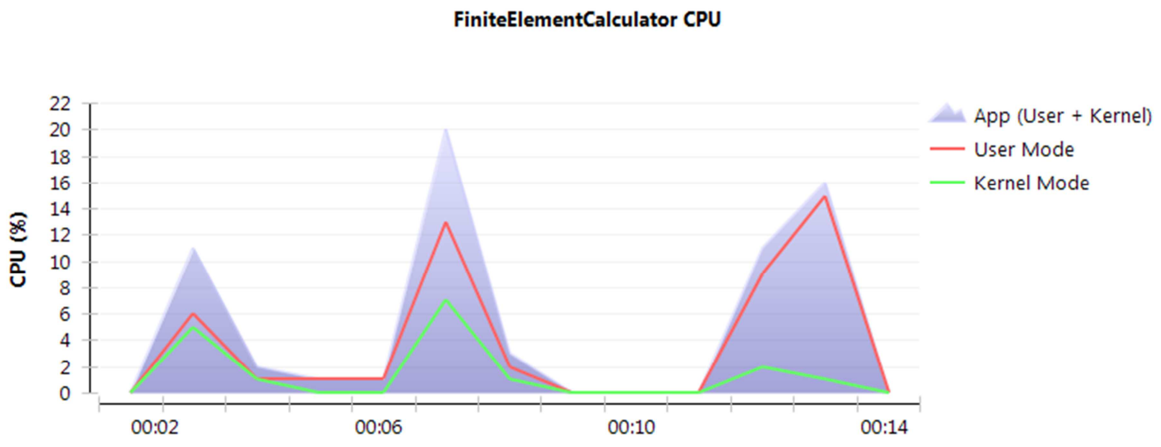
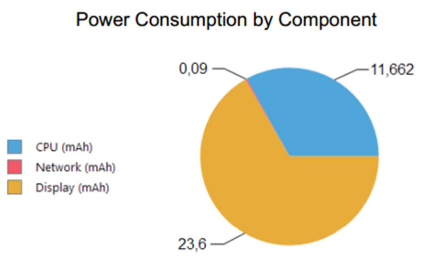
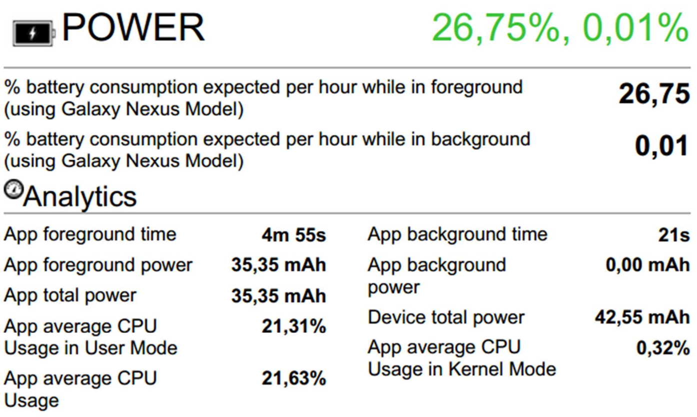


Figure 4. FiniteElementCalculator CPU load while using the cloud

The pie diagrams (figure 5-6) display how power consumption has altered. Both runs are based on the same test scenarios.





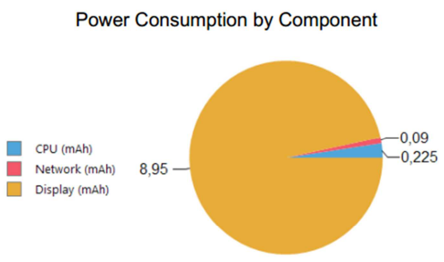
POWER 26,75%, 0,01%

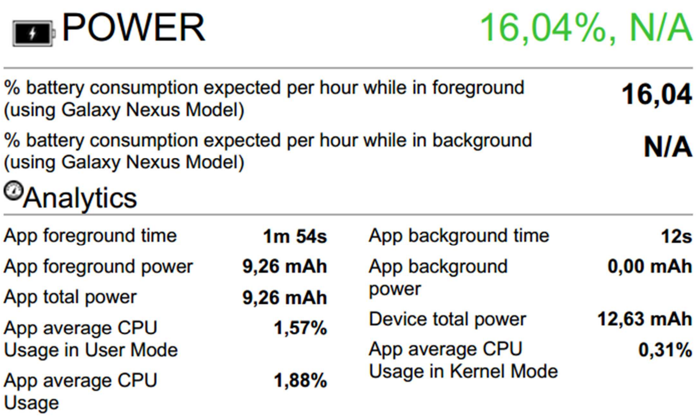
% battery consumption expected per hour while in foreground (using Galaxy Nexus Model)	26,75
% battery consumption expected per hour while in background (using Galaxy Nexus Model)	0,01

Analytics

App foreground time	4m 55s	App background time	21s
App foreground power	35,35 mAh	App background power	0,00 mAh
App total power	35,35 mAh	Device total power	42,55 mAh
App average CPU Usage in User Mode	21,31%	App average CPU Usage in Kernel Mode	0,32%
App average CPU Usage	21,63%		

Picture 5. FiniteElementCalculator power consumption per component while running on a mobile device





POWER 16,04%, N/A

% battery consumption expected per hour while in foreground (using Galaxy Nexus Model)	16,04
% battery consumption expected per hour while in background (using Galaxy Nexus Model)	N/A

Analytics

App foreground time	1m 54s	App background time	12s
App foreground power	9,26 mAh	App background power	0,00 mAh
App total power	9,26 mAh	Device total power	12,63 mAh
App average CPU Usage in User Mode	1,57%	App average CPU Usage in Kernel Mode	0,31%
App average CPU Usage	1,88%		

Picture 6. FiniteElementCalculator power consumption per component while using the cloud

It is also shown, that it takes less time to execute the tasks remotely. So in this case, migrating functionality to the cloud not only decreased energy consumption on the mobile device it also decreased calculation speed.

AsciiCamImage converts an image into an ASCII picture. The *AsciiCamImage* example has revealed some restrictions on the approach. Migration to the cloud required the adaptation of platform specific libraries. Consequently, the *AsciiCamImage* implementation referred to Dalvik-specific code (that is, the code depending on the Dalvik virtual machine), which had to be adapted to the JavaVM used on the server. Exchanging data with JSON (JavaScript Object Notation) [9] also loaded the CPU more intensively, when large data was to be transferred. In that setting it was not possible to reach the key conditions for reducing energy consumption. Consequently no power was saved (cf. Table 2). Moreover, this experiment also showed that slow connection speed also effectuates an unacceptable performance. The challenges faced while working with this application are summarized in "Lessons learned" section.

The *AsciiCamVideo* application is an extensive modification of AsciiCamImage which converts video files to videos showing the appropriate ASCII pictures: the initial application was improved in order to divide an input video into the frames, process each frame into an ASCII picture, and, finally, build a new video out of these ASCII frames. In contrast to the previously sketched simple AsciiCamImage migration to the cloud, both, energy consumption and performance were improved

	Local			Fast connection (D 6-12 Mbps/ U 7,7-12)			Slow connection (D 0.9-2.4 Mbps/ U 0,6-0,8)			
	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Data volume (out/in), Mb
AsciiCam Image										
Heavy image										5487.25 / 150.86
Average	22.5	33	32	26.3	45.4	116.8				
Standard deviation	2.54	2.74	5.39	4.18	4.37	62.95				
Medium image							13.69	18.95	800	2887.77 / 68.8
Average	18.4	29.8	26	28.1	41.9	36.7				
Standard deviation	2.94	5.2	5.29	0.51	1.08	3.79				
Light image										430.67 / 116.23
Average	18.4	25.2	17	24	32.3	20.7	14.5	37	230	
Standard deviation	2.94	3.03	1	1.38	2.23	1.53	0.28	1.15	7.55	

Table 2. AsciiCam Image power measurement results

when processing video files. The power measurement results (Table 3) show that using the cloud allows to gain more than two times decrease in energy consumption. The reason is that video files are sent to the server as they are, instead of writing and reading JSON. Not using JSON does not load the CPU significantly, and consequently power was saved, although the amount of data is bigger than in the case with pictures.

	Local			Fast connection (D 6-12 Mbps/ U 7,7-12)			Slow connection (D 0.9-2.4 Mbps/ U 0,6-0,8)			
	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Battery, %	CPU, %	Time, s	Data volume (out/in), Mb
AsciiCam Video										
Short video										2.73 / 30.26
Average	34.8	47.7	1260	16.6	38.5	224.7	16.3	40.7	809	
Standard deviation	3.2	5.3	129.62	2.75	0.79	53	0.87	1.52	188	

Table 3. AsciiCam Video power measurement results

Summarizing, it can be seen, that for all successful experiments on migrating functionality to the cloud, it was possible to reduce energy consumption by 30 - 54% and run time by 25 - 83%.

3. Lessons Learned

Before moving the code to the cloud, it's worth to estimate:

- the ratio of time to transmit the data over the network to time needed to process the data on a device. If the first is much longer, probably, it does not make sense to migrate tasks to the cloud;
- the additional CPU load which appears when we want to execute the task remotely (for example, writing data to JSON before sending), as it may be very resource-hungry;
- the CPU load while executing the task on a device. If the CPU speed is low and required time is little, probably there are no benefits when migrate the task to the cloud.

Based on these experiences code should not or cannot be migrated to the cloud towards improving energy consumption:

- if the required changes modify the applications state without returning the new value,
- if the code uses operating system or platform-specific classes and stubs,
- if the code uses native code (that needs to be recompiled or licensed for the server).

As for native code, applications using Android NDK "*will be more complicated, have reduced compatibility, have no access to framework APIs, and be harder to debug*" [13] Using native code does not necessarily result in performance improvements [6] and it may be sensible to refrain from using it.

Acquiring data from the system services such as getting a GPS signal or processing sensor updates is very power hungry, but mostly cannot be managed in the cloud without the device's hardware. However, it is claimed that new sensors introduces with Android 4.4 KitKat are more reserved concerning energy consumption [2]

Energy consumption of applications in cloud can be different, depending of hardware devices and the structure of load of processing resources. But almost all devices in remote datacenters, used for cloud computing, satisfy to Energy Star standard from EPA [14], used for technologies with advanced energy efficiency. Furthermore, moving more functionality to data centers also allows for more energy optimization and better load balancing. Using virtualization techniques and switching of not used (hardware) servers, will decrease energy consumption in data centers (cf. e.g. [7]). Probably, even from a global point of view, energy might be saved, when migrating functionality to the cloud. This requires more in depth analysis, not shown in the small experiments, presented here.

4. Related Work

Benefits on remote code execution towards power saving were studied in several works.

Jason Flinn [4] implements local, remote and hybrid applications' execution and shows their effects on energy savings. The author refers to 30% savings when using remote execution. The experiments presented in this study achieves up to 54% reduction.

Miettinen and Nurminen [11] analyze factors affecting energy consumption of mobile clients in cloud computing. They present the measurements about the central characteristics of contemporary mobile devices that define the basic balance between local and remote computing.

Abdelmotalib and Wu [1] explore the energy consumption of mobile devices' components, such as CPU, display, wireless interfaces and others. This information serves as a basis of this work.

Moreover, the presented approach applied reengineering techniques, and provided recommendations for the developers doing the required migrations. Moving functionality from an integrated system to a distributed system using cloud services requires intensive architectural migration activities ensuring the migrations quality. The used migration process (cf. [5]) should consider, clearly identifying code fragments to be migrated, and deciding on the most suited migration strategy leading to measurable energy savings and clear estimations on changes in performance.

5. Conclusion

Basing on the experiments of this work, only code

- not modifying the state of the application,
- not using Android-specific resources, such as UI or system services,
- not using platform specific classes and stubs,
- not using native code (or needs to be recompiled for the target server)

could be migrated without additional effort.

The experiments showed that delegating parts of mobile app’s functionality to the cloud can save battery power on mobile devices.

Firstly, applying a migration approach saves energy in many cases. This is achieved by reducing the CPU load, as a device now does not have to compute all the results by itself, but this function is delegated to the cloud. The amount of energy required to send data to and receive it from the server is usually less than the power needed to execute these methods on a device itself. But there can be exceptions. For example, if you have very big data to transfer by JSON strings, then you are probably going to increase the device's CPU load and therefore energy will not be saved. Thus, you should consider adjusting the way to transfer data to its amount and type.

Secondly, each application accomplishes some functionality which requires a lot of resources. These blocks of code can be extracted and ported to the cloud to be executed there. This process is can be divided into some particular steps (described in chapters 3 and 5), but a number of challenges can be faced while doing this. Some code could be closely coupled with the mobile platform, so it may occur to be troublesome to run it on another one. Depending on the quality of the source system, it also can be difficult to identify those code fragments forming a coherent functionality to be extracted and migrated. The example systems investigated here, were clearly structured, so this issue did not rise.

Thirdly, the performance (time) of the app can change. The performance depends on the apps complexity, the amount of data to be exchanged between the device and the server, and the speed of internet connection. That is why you should test your application in order to decide whether the performance variations are acceptable for a user. It may be sensible to provide a timeout to run a task locally if there is no answer from the cloud.

References

[1] Abdelmotalib A., Wu Z. "*Power Consumption in Smartphones (Hardware Behaviourism)*", *http://ijcsi.org/papers/IJCSI-9-3-3-161-164.pdf*, (download: 13.11.2013), 2012.

[2] Android Developers "*Android 4.4 KitKat and Updated Developer Tools*", http://android-developers.blogspot.de/2013/10/android-44-kitkat-and-updated-developer.html (download: 13.01.2014), 2013.

[3] Dozing Cat Software "*AsciiCam*", https://play.google.com/store/apps/details?id=com.dozingcatsoftware.asciicam (download: 26.11.2013).

[4] Flinn J. "*Extending Mobile Computer Battery Life through Energy-Aware Adaptation*", PhD thesis, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. http://notrump.eecs.umich.edu/papers/thesis.pdf (download: 25.07.2014), 2001.

[5] Fuhr, A.; Winter, A.; Erdmenger, U.; Horn, T.; Kaiser, U.; Riediger, V.; Teppe, W. "*Model-Driven Software Migration - Process Model, Tool Support and Application*", In: Ionita, A.; Litoiu, M.; Lewis, G. eds. "*Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments"*, Hershey, PA, IGI Global, pp. 153-184, November 2012.

[6] Google "*Android NDK*", http://developer.android.com/tools/sdk/ndk/ (download: 08.12.2013).

[7] Hoyer, M; Schröder, K.; Schlitt, D.; Nebel, W. „*Proactive dynamic resource management in virtualized data centers*", in Proceeding e-Energy '11 Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking, ACM pp 11-20, 2011.

[8] Hruska, J. "*The future of CPU scaling: Exploring options on the cutting edge*" http://www.extremetech.com/extreme/120353-the-future-of-cpu-scaling-exploring-options-on-the-cutting-edge (download: 13.11.2013), 2012.

[9] JavaScript Object Notation "*Introducing JSON*", http://www.json.org/ (download: 24.07.2014).

[10] Little Eye Labs "*How Little Eye Measures Power Consumption*" http://www.littleeye.co/blog/2013/07/30/how-little-eye-measures-power-consumption/ (download 22.01.2014).

[11] Miettinen A., Nurminen J. "JSON*Energy efficiency of mobile clients in cloud computing*", https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Miettinen.pdf (download: 13.11.2013).

[12] Strokova V. "*Cloud computing for mobile devices: Reducing energy consumption*", Master Thesis, University of Oldenburg, Software Engineering Group, http://www.se.uni-oldenburg.de/documents/strokova-MA2014.pdf (download: 24.07.2014), 2014.

[13] Turner D. "*Introducing Android 1.5 NDK, Release 1*", http://android-developers.blogspot.de/2009/06/introducing-android-15-ndk-release-1.html, (download: 07.12.2013), 25.06.2009.

[14] United States Environment Protection Agency "*Energy Star*" http://www.energystar.gov (download: 25.07.2014).