SmartDiet: Offloading Popular Apps to Save Energy

Aki Saarinen, Matti Siekkinen, Yu Xiao, Jukka K. Nurminen, Matti Kemppainen
Aalto University, School of Science, Finland
aki@akisaarinen.fi, {matti.siekkinen, yu.xiao, jukka.k.nurminen}@aalto.fi,
matti.kemppainen@iki.fi

Pan Hui
Deutsche Telekom Labs, Berlin, Germany
pan.hui@telekom.de

ABSTRACT

Offloading computation to cloud has been widely used for extending battery life of mobile devices. However, little effort has been invested in applying the offloading techniques to communication-related tasks. We propose SmartDiet, a toolkit to identify the constraints that reduce offloading opportunities and to calculate the energy-saving potential of offloading communication-related tasks. SmartDiet traces the method-level application execution and estimates the allocation of communication energy cost from traffic traces. We discuss key features of SmartDiet and show some preliminary results using a prototype implementation.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

Offloading, energy consumption, constraint analysis

1. INTRODUCTION

A seemingly straightforward way to conserve battery life of a mobile device is to reduce workload by migrating the whole or part of the application execution to a more powerful machine called *surrogate*. This method is called *offloading* or sometimes *cyber foraging*. Several frameworks including MAUI [3], Cuckoo [4], CloneCloud [2], ThinkAir [5], and Scavenger [6] have been used for implementing computation offloading that aims at reducing energy cost by CPU and memory. Although most popular apps today involve intensive communications that consume a significant part of the overall energy consumption on mobile devices, little effort has been put on offloading communication-related tasks and it is unclear whether and in which way such offloading can save energy. To address this issue, we investigate the feasibility of such offloading in this work, taking a set of open source applications as examples. We refer to such offloading as *communication offloading*.

There are two means by which energy savings could be achieved from communication offloading. First is to reduce the amount of the network traffic that needs to be handled by the mobile device. For example, signaling traffic such as keep-alive messages can be partly suppressed. Second is to optimize the pattern shown in the traffic and to improve the overall latency and/or throughput. For instance, packet interval pattern and throughput have a significant impact on communication energy cost[9].

In this study we take the first look into the feasibility of communication offloading through case studies. We use a specific method-level application partitioning framework called ThinkAir [5]. In general, existing frameworks for method-level application partitioning provide APIs for specifying which methods can be possibly offloaded, while requiring programmers with expert knowledge to manually annotate these methods. To better utilize these frameworks, tools that can automatic the partitioning are desperately needed.

The methods that can be marked as offloadable should first be suitable for remote execution. In practice, there are many constraints that limit the remote execution of certain methods. These constraints are non-trivial and laborous to identify manually. Based on case studies of open source apps, we summarize the constraints into three types and develop a toolkit called SmartDiet for identifying such constraints automatically. Our toolkit also provides suggestion of code modifications for releasing certain constraints such as serialization issue in method-level offloading. Furthermore, as communication cost is heavily depedent on traffic pattern, traffic pattern can be shaped through code reconstruction in order to reduce communication cost. To assist developers in making such improvement, our toolkit provides energy estimation at method level. It can be used for evaluating the energy-efficiency and performance of code reconstruction at development stage, and can therefore guide programmers to improve application implementation for better energy-efficiency.

2. SMARTDIET

Based on our experiences in offloading open-sourced popular apps, described in [8], we conclude that for offloading to be feasible in practice, further assistance for developers in profiling and modifying existing programs is needed. We share many concerns that Balan et al. have presented in [1]. Their goal was to enable rapid modification of applications for cyber foraging so that the developer first creates a so-called tactics file corresponding to the program being modified, after which the actual program code is modified. This is still a pretty laboursome process. Our approach is to provide the SmartDiet toolkit to help developers who are using

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
ACM 978-1-4503-1419-0/12/08.

Table 1: Constraint statistics for methods within 16 open source apps.

Statistic	Median	Min	Max
Number of methods	431	121	4411
Directly migratable	0.17%	0.00%	3.70%
Migratable with minor changes	15.7%	0.00%	46.8%
Hardware access constraints	14.2%	2.28%	41.3%
Potential unexpected behavior because of access to file system	10.7%	0.00%	30.3%

highly automated offloading frameworks such as ThinkAir and MAUI. SmartDiet comprises two tools: energy profiling and constraint identification. We have designed and implemented prototypes of both. Detailed descriptions are in [8].

Energy profiling tool: This tool finds and visualizes parts of application code that could yield energy savings if offloaded. The tool collects two kinds of information while the program is running: the traffic trace, and a trace of the program execution flow to later produce class and method level statistics for the developer. We use statistical methods to match each collected network packet into one single method call in the program. We apply power models [9] to get detailed time series of energy consumption of the wireless network interface in use. Similar to Eprof [7], the models we use take tail power state into account. Moreover, our power models account for the impact of power consumption differences in each active state during the data transmission. Program execution in each thread can be viewed as a hierarchical call tree, where a method calls another method which calls another and so on. Our tool reconstructs this tree, carrying along the information of the detected network usage. It then aggregates the traffic of the nodes up in the tree, so that the root method, where the execution starts, gets associated with all packets that have been sent or received within each thread.

Constraint identification tool: This tool identifies offloading constraints through static analysis of the source code, determines which methods can be offloaded as such and points out trouble spots in the code. The developer would apply this tool after first profiling the energy consumption of the app to identify the candidate methods for offloading. For each method of the application, it points out problems that can prevent offloading unless the code is modified. The tool currently uses heuristics that identify problems associated with Android platform and Java Serialization API which is used to implement the remote execution of methods. Similar heuristics can be crafted to other remote execution mechanisms, e.g., the Android Parcelable mechanism or .NET serialization.

Hardware constraints: The first set of constrained methods are those that require access to the hardware of the local device. We currently identify method as having this constraint if it tries to show, for instance, notifications to the user, update anything on the screen, vibrate the phone, access the Bluetooth, wifi or usb subsystem, and so on. We identified 20 constrained subsystems while going through Android system APIs.

We ran our constraint analysis tool with a set of programs that are non-trivial in size and include either communication or non-trivial computation. Results are shown in Table 1. Maximum of 3% of methods are directly migratable. SmartDiet can improve the situation by pointing trouble spots in the source code. If all of these issues were fixed, it would enable the migration of 15% to 47% of methods. SmartDiet can also guide the developer into fixing the issues regarding hardware or file system access. The results show that typical existing apps are heavily constrained in terms of what can be offloaded but with a little assistance, developers could leverage offloading much more in practice.

3. CONCLUSIONS AND FUTURE WORK

SmartDiet has shown promising results in helping the development of offloadable code, although some challenges still remain to complete all the features we believe would be useful. We will integrate energy estimator of CPU usage to the same toolkit using existing models. We also study ways to statistically analyze the dependencies between execution of multiple threads and to track packet flow inside a program. Programming styles and application structures are also interesting topics for future work.

4. ACKNOWLEDGMENTS

This work was supported by the Academy of Finland, grant number 253860.

5. REFERENCES

[1] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *MobiSys 2007*, pages 272–285.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *EuroSys 2011*, pages 301–314.

[3] 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 *MobiSys 2010*, pages 49–62.

[4] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, volume 76, pages 59–79. Springer Berlin Heidelberg, 2012.

[5] 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 2012*, pages 945–953.

[6] M. Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *PerCom 2010*, pages 217–226.

[7] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *EuroSys 2012*, pages 29–42.

[8] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui. Offloadable apps using smartdiet: Towards an analysis toolkit for mobile application developers. *CoRR*, abs/1111.3806, 2011.

[9] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *e-Energy 2010*, pages 75–84.