The Journal of Systems and Software 117 (2016) 307–316

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

From benchmarks to real apps: Exploring the energy impacts of performance-directed changes

Cagri Sahin*, Lori Pollock, James Clause

Computer and Information Sciences Department, University of Delaware, Newark, DE 19716, USA

ARTICLE INFO

Article history:
Received 7 November 2015
Revised 9 February 2016
Accepted 10 March 2016
Available online 18 March 2016

Keywords:
Android applications
Performance tips
Energy efficiency

ABSTRACT

Battery life is an increasing concern for mobile devices. Recent studies have provided initial evidence that applying performance tips is an effective mechanism for decreasing energy usage. However, the generalizability of such studies to real applications is unclear. We aim to provide deeper insights into whether mobile application developers can effectively reduce the energy consumption of their applications by applying performance tips.

We conducted an empirical study to investigate the energy impacts of applying four commonly suggested performance tips to eight real Android applications. Considered performance tips are unlikely to impact energy usage in a statistically significant manner and, even when the impacts are statistically significant, the change in battery life is around 1%. Mobile application developers cannot expect to improve the energy usage of their applications as a by product of performance improvements. Tools and techniques that specifically target energy usage are necessary for significant improvements.

1. Introduction

The types of computing devices used by consumers and enterprises have shifted significantly over the past decade. Whereas in 2010, traditional PC sales (desktops and laptops) outnumbered other computing platforms, in 2014, roughly 318 million PCs were sold compared to 216 million tablets and 1.3 billion smart phones (Gartner, 2015). With the increased use of mobile devices has come increased concerns about the amount of energy that they consume. In particular, battery life is critical to mobile devices. Extending a device's battery lifetime is now a priority as excessive battery drain is a major contributor to users giving poor reviews or requesting refunds (Apigee, 2012).

Unfortunately, software engineers have a difficult time writing energy-efficient software, and they are not as successful as they could be, because they do not understand how the decisions they make affect the energy usage of their applications. To combat the lack of knowledge available to mobile application developers, researchers have begun to investigate how changes they make to their software impact their software's energy consumption. These investigations include empirical studies on the impacts of applying code obfuscations (Sahin et al., 2014b), performance tips (Li and Halfond, 2014; Mundody and K, 2014; Tonini et al., 2013), and ad blockers (Rasmussen et al., 2014); identifying energy expensive APIs (Linares-Vasquez et al., 2014); and investigating how the energy usage of applications changes over time (Hindle, 2012).

Some studies have provided initial evidence that applying performance tips—best practices oriented toward runtime performance—is an effective mechanism for decreasing energy usage. More specifically, Li and Halfond (2014), Tonini et al. (2013), and Mundody and K (2014) all report that applying performance tips can decrease energy usage from 10% to 67% for Android programs. This is promising because such tips are both easy to understand and easy to apply. In addition, these results support the common wisdom that applications can save energy by "racing to sleep"—speeding up computation to allow the CPU to reach a low power state faster. These results also show that performance tips are related to energy code smells in that energy code smells are implementation choices at source code level that cause higher energy consumption (Vetro et al., 2013). Consequently, performance tips are potentially more likely to be used in practice. However, these studies are limited in scope in several ways. The most severe of these limitations is that none of the existing studies evaluated the impacts of the performance tips when applied to real applications. Rather, they applied the performance tips to kernels or micro-benchmarks—small pieces of code that focus on the specific issue under study. While the targeted nature of kernels is beneficial, it remains unclear whether the observed results will transfer to real applications, which are characteristically larger and more complex.

* Corresponding author. Tel.: +1 3028316339.
E-mail address: cagri@udel.edu (C. Sahin).

http://dx.doi.org/10.1016/j.jss.2016.03.031
0164-1212/

To better understand the energy impacts of performance tips on Android applications, we have conducted an empirical study investigating the energy impacts of applying four commonly recommended performance tips to real Android applications. This study provides deeper insight into whether Android application developers can effectively reduce the energy consumption of their applications by applying performance tips. At a high-level, the study is focused on addressing two major questions: (1) How do performance tips alter the overall energy consumption of Android applications?, and (2) Are the energy impacts of performance tips likely to be meaningful for Android application users?

To answer our research questions, we created a total of 32 modified versions of eight Android applications by applying the four considered performance tips. We executed both the modified and base versions of the applications multiple times on two different platforms with one or more user scenarios. As the versions were executing, we recorded the total amount of energy that was consumed. In total, we ran 5100 executions, 2550 per platform, and collected over 3GB of experimental data. We then performed a statistical analysis of the collected data to investigate the impacts of the performance tips on energy usage and to answer our research questions.

The results of our study show that:

(1) Despite initial evidence to the contrary, the performance tips that we investigated are unlikely to impact the energy usage of Android applications; only 2% of the 136 cases in our study indicated a statistically significant difference in energy consumption.
(2) Even in the unlikely event that a performance tip impacts energy usage in a statistically significant manner, the impact of change on battery life is negligible; the largest percentage change in battery life that we observed was approximately ≈1%.

The remainder of this paper is organized as follows: Section 2 describes the methodology of our study including our subjects and experimental procedure. Section 3 presents and discusses the results of the study including potential threats to its validity. Finally, Sections 5 and 6 discuss related work and present our conclusions and future work.

2. Empirical study

This section describes the details of our study design, including our independent and dependent variables; considered applications, scenarios, and performance tips; measurement platforms; and data collection protocol. In planning this work, we followed a methodology that is nearly identical to the one used in our prior work on investigating the impacts of code obfuscation on energy usage (Sahin et al., 2014b). To design this methodology, we followed well-known guidelines for empirical study design (Arcuri and Briand, 2011) and our experience from conducting similar studies (Sahin et al., 2012; Manotas et al., 2013; Sahin et al., 2014a). All of our experimental applications, artifacts, and summary data are publicly available: https://bitbucket.org/udse/perf_tips-study. Our raw experimental data is too large to host publicly, but is available upon request.

2.1. Experimental variables

In this study, we considered one dependent variable, the amount of energy consumed by an execution, and two independent variables: (1) the performance tip applied to the application, and (2) the platform where the application executes.

To isolate the impacts of changing our independent variables on our dependent variable, it is necessary to precisely control how the applications are executed. In general, mobile applications are interactive and event-driven. They accept input, either from a user or from a sensor, perform some computation, and generate a response. In our experiments, this interactive nature can introduce a potential source of bias as it is difficult to manually reproduce a given execution exactly. For example, a user can often repeatedly perform the same sequence of actions (e.g., enter text into a textbox or click a button) but cannot maintain the same timing between the actions. Although such differences may seem inconsequential, they may lead to observed differences in energy consumption that are not due to changing our independent variable, but rather to differences in how the application is driven. To prevent such bias, it is necessary to be able to reproduce deterministically a given sequence of actions with great fidelity. Capture/replay tools provide this functionality.

Capture/replay tools are designed to allow for the deterministic replay of a sequence of recorded events. Conceptually, this is accomplished by wrapping an application to insulate it from its environment. When capturing interactions, the wrapper records all of the events that are passed to the application from the environment. During replay, the wrapper replaces the environment and passes the recorded events to the application. Because precise timing information is recorded during the capture process, there is very little variability in when events are passed to the application during replay. Hence, when using a capture/replay tool, any observed variations in energy usage are more likely to be the result of the performance tips applied rather than inconsistencies in driving the application. We chose to use RERAN (Gomez et al., 2013) as our capture/replay tool, because it is designed to record and replay Android applications.

Table 1
Considered applications.

Application	Description	LoC
Calculator	Android calculator	1427
Clock	Android clock	13477
DailyMoney	Daily financial tracker	8723
Nim	Strategy game	1475
OIFileManager	File manager	7200
OpenSudoku	Sudoku game	6079
SkyMap	Astronomy application	10921
Tomdroid	Note taking application	7955

2.2. Considered applications

For our study, we used popular, easily accessible Android applications as our subjects. We selected Android applications for several reasons. First, as is the case for most software engineers, Android developers often care about the performance of their applications. As such, there are numerous performance tips that have been suggested for Android applications. Second, as mentioned previously, Android application developers typically care about the energy efficiency of their applications. Third, the source code of many Android applications is freely available, allowing us to easily modify the applications to apply the performance tips. Finally, we have extensive infrastructure to run Android applications and measure their energy usage.

Table 1 lists the specific applications that we used in this study. The first two columns, *Application* and *Description*, list the name of each application and a brief description of its functionality, respectively and the final column, *LoC*, shows the application's number of lines of code.

We chose these specific applications for several reasons. First, they are representative of a wide variety of common application types (e.g., games, study aids, productivity tools). Second, they are