International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

A study of Automated Energy Optimization for Android Phone

Jayashri Khedkar[1]; GaneshHiwale[2]; SonaliMukhekar[3]&S. Pratap Singh[4]
BE Computer Department, SP's IOK COE, PUNE[1, 2, 3]
Prof. Computer Department, SP's IOK COE, PUNE[4]
Sweetjaya1011@gmail.com[1], ganesh_hivale2011@rediffmail.com[2],
mukhekarsonali29@gmail.com[3],pratap.singh.s@gmail.com[4].

ABSTRACT:

The Smartphone application market is growing rapidly. Many Android applications are not energy efficient. Locating energy problems in Android applications is difficult. GreenDroid is automated approach to diagnosing energy problems in Android applications. GreenDroid will also use to diagnosis the energy problems in mobile sensors like GPS, Wi-Fi, Bluetooth, Light Sensor etc.It monitors sensors and wake lock operations to detect missing deactivation of sensors and wake locks. It also tracks transformation and usage of sensory data and judges whether they are effectively utilized by the application using our state sensitive data utilization metric.GreenDroid is evaluated using Android applications and mobile sensors.Our research paper consists of comprehensive study of additional use of GreenDroid. It can be used to optimize energy usage automatically. This approach can generate detailed reports with actionable information to assist developers in validating detected energy problem. We can develop an Android application which is automatically turned off the mobile sensors which are not in current use.

Keywords: Smartphone Applications; Energy problems; mobile sensors; automated diagnosis; Green Droid

1. INTRODUCTION:

Smartphone has become pervasive. Smartphone are as powerful as the PCs. Therefore, they are perfectly suitable to becomethe first real-life platforms for ubiquitouscomputing.Sensing operations are usually energy consumptive,and limited battery capacity always restricts such anapplication's usage. As such, energy efficiency becomes acritical concern for Smartphone users. Existing studies show that many Android applications arenot energy efficient due to two major reasons [1]. First, theAndroid framework exposes hardware operation APIs (e.g.APIs for controlling screen brightness) to developers.Although these APIs provide flexibility, developers have to beresponsible for using them cautiously because hardware misusecould easily lead to unexpectedly large energy waste.Second, Android applications are mostly developed by smallteamswithout dedicated quality assurance efforts. Their developersrarely exercise due diligence in assuring energy savings. Locating energy problems in Android applications is difficult.

By examining bug reports, commitlogs, bug-fixing patches, patch reviews and release logs of Android applications,we made an interesting observation: Although the root causesof energy problems can vary with different applications, many ofthem are closely related to two types of problematiccoding phenomena:

Missing sensor or wake lock deactivation: To use a Smartphonesensor, an application needs to register a listenerwith the Android OS. The listener should be unregisteredwhen the concerned sensor is no longer being used. Similarly,to make a phone stay awake for computation, anapplication has to acquire a

wake lock from the Android OS. The acquired wake lock should also be released as soonas the computation completes. Forgetting to unregister sensorlisteners or release wake locks could quickly deplete afully charged phone battery [2] [3].

Sensory data underutilization: Smartphone sensors probetheir environments and collect sensory data. These data areobtained at high energy cost and therefore should be utilizedeffectively by applications. Poor sensory data utilization canalso result in energy waste. For example, Osmdroid, a popular navigation application, may continually collect GPS datasimply to render an invisible map. This problem occursoccasionally at certain application states. Battery energy isthus consumed, but collected GPS data fail to produce anyobservable user benefits.

To realizeefficient and effective analysis of sensory data and data utilization by applications in Smartphone, we need to address tworesearch issues and two major technical issues as follows[8].

Research issues: While existing techniques can be adaptedto monitor sensor and wake lock operations to detect their missing deactivation, how to effectively identify energy problems arising from ineffective uses of sensory data is anoutstanding challenge, which requires addressing tworesearch issues. First, sensory data, once received by anapplication, would be transformed into various forms andused by different application components. Identifying programdata that depend on these sensory data typicallyrequires instrumentation of additional code to the originalprograms. Manual instrumentation is undesirable because it is labor-intensive and error-prone. Second, even if a programcould be carefully instrumented, there is still no well-definedmetric for judging ineffective utilization of sensory dataautomatically. To address these research issues, we proposeto monitor an application's execution and perform dynamicdata flow analysis at a bytecode instruction level. This allowssensory data usage to be continuously tracked without any need for instrumenting the concerned programs. We alsopropose a state-sensitive metric to enable automated analysisof sensory data utilization and identify those applicationstates whose sensory data have been underutilized.

Technical issues: JPF was originally designed for analyzingconventional Java programs with explicit control flows. Itexecutes the byte code of a target Java program in its virtualmachine. However, Android applications are event-drivenand depend greatly on user interactions. Their program codecomprises many loosely coupled event handlers, amongwhich no explicit control flow is specified. At runtime, theseevent handlers are called by the Android framework, whichbuilds on hundreds of native library classes.As such, applying JPF to analyze Android applications requires: (1) generatingvalid user interaction events, and (2) correctly schedulingevent handlers. To address the first technical issue, we proposeto analyze an Android application's GUI layout configurationfiles, and systematically enumerate all possible userinteraction event sequenceswith a bounded length at runtime.We show that such a bounded length does not impair the effectiveness of our analysis, but instead helps quickly exploredifferent application states and identify energy problems. Toaddress the second technical issue, we present an applicationexecution model (AEM) derived from Android specifications.This model captures application-generic temporal rules thatspecify calling relationships between event handlers. Withthis model,we are able to ensure an Android application to beexercised with correct control flows, rather than being

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

randomlyscheduled on its event handlers. As we will show inour later evaluation, the latter brings almost no benefit to theidentification of energy problems inAndroid applications.

2. BACKGROUND

We select the Android platform for our study because it is currently one of the most widely adopted Smartphone platforms and it is open for research. Applications running on Android are primarily written in Java programming language.An Android application is first compiled to Java virtualmachine compatible .class files that contain Javabytecode instructions. These .class files are then convertedto Dalvik virtual machine executable .dex files that containDalvik bytecode instructions. Finally, the .dex files areencapsulated into an Android application package file (i.e. an .apk file) for distribution and installation. For ease of presentation,we in the following may simply refer to "Androidapplication" by "application" when there is no ambiguity.An Android application typically comprises four kinds of components as follows [7]:

Activities: Activities are the only components that allowgraphical user interfaces (GUIs). An application may usemultiple activities to provide cohesive user experiences.The GUI layout of each activity component is specified inthe activity's layout configuration file.

Services: Services are components that run at backgroundfor conducting long-running tasks like sensor data reading.Activities can start and interact with services.

Broadcast receivers: Broadcast receivers define how anapplication responds to system-wide broadcasted messages.It can be statically registered in an application's configurationfile (i.e., the AndroidManifest.xml file associated witheach application), or dynamically registered at runtime bycalling certain Android library APIs.

Content providers: Content providers manage sharedapplication data, and provide an interface for other componentsor applications to query or modify these data.

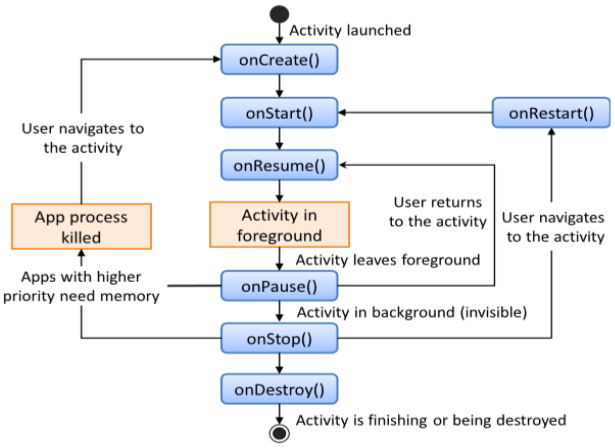

Fig1: An activity's lifecycle diagram

3. EXISTING TECHNOLOGIES

Cloud Offloading: Cloud offloading is a mobile application optimization technique that makes it possible to execute the application's energy intensive functionality in the cloud, without draining the mobile device's battery. Cloud offloading is typically implemented as a program partitioning transformation that splits a mobile application into two parts: client running on a mobile device and server running in the cloud; all the communication between the parts is conducted via a middleware mechanism such as XML-RPC. Thus, cloud offloading is a special case of automated program partitioning-distributing a centralized program to run across the network using a compiler-based tool transform a centralized program or migrating execution between different application images at the OS level. The promise of cloud offloading is demonstrated by the proliferation of competing approaches to this technique in the literature. We partition applications without destroying their ability to execute locally. All of

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

the prior cloud offloading techniques shares the goal of reducing the energy consumed by mobile devices. The approach presented in this paper adopts many of the techniques above to automatically transform mobile applications without any changes to their source code and to synchronize program states between partitions. However, our approach's goal is to improve on the efficiency of the prior cloud offloading technique by postponing the offloading decisions until the runtime, when a feedback-loop mechanism can determine which amount of offloading is optimal.

Energy Consumption Patterns in Mobile Applications: Network communication constitutes one of the largest sources of energy consumption in a mobile application. According to a recent study, network communication consumes between 10 and 50% of the total energy budget of a typical mobile application. Specifically, in our prior research, we measured and analysed how middleware can significantly affect a mobile application's energy consumption. Our experiments assessed the energy consumption of passing varying volumes of data over networks with different latency/bandwidth characteristics. Then, we isolated how mobile applications consume energy to infer their common energy consumption patterns. The experimental results and systematic analysis conducted through that research inspired us to initiate the work presented in this paper.

Program Analysis: Program analysis codifies a set of techniques to infer various facts about the source code to be leveraged for optimization and transformation. Class hierarchy analysis (CHA) constructs a call graph in object-oriented languages. Dataflow analysis determines how program variables are assigned to each other. Side-effect free analysis determines whether a method changes the program's heap. CHA is used to compute the functionality to offload and the program state to transfer for a given offloading. To select optimal offloading strategies, we combine dataflow and side-effect analyses. Based on the results, a bytecode enhancer then rewrites the application without changing its source code. It used Soot to implement our program analysis and transformations.

4. PROPOSED WORK:

GreenDroid will be used as an energy optimizer in Smartphone. It first list out all available sensors of Smartphone. After that analyses the sensory data and its utilization. Applications current state and related data utilization from the sensor will used as a input data for GreenDroid. Working on Wi-Fi and Bluetooth will depend on data under utilization of Wi-Fi and data transaction information of Bluetooth respectively. Analysis report contains the current state and data utilization of application and sensors. Utilized and unused sensors will be disabled. Appropriate action will be taken by GreenDroid as per analysis report. As per that approach we can save the unwanted battery consumption. Hence it optimizes the energy for Android phones.

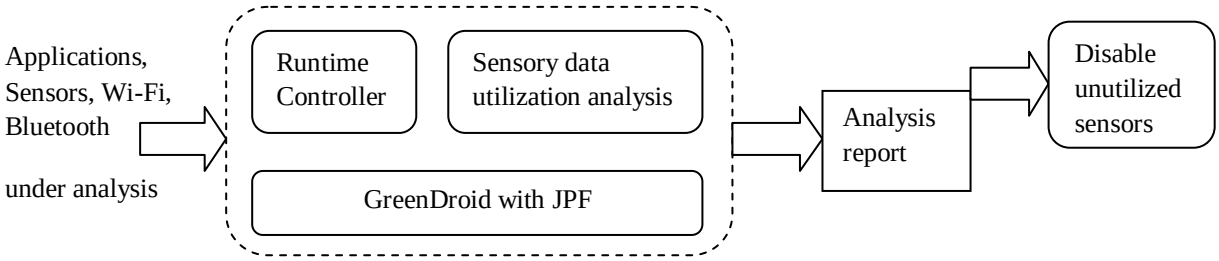

Fig2: Approach Overview

Using this approach we can optimize mobile battery consumption using GreenDroid. In this approach automatic diagnosis and optimization will done sequentially. In future, we will study more efficient energy optimization techniques and use of GreenDroid. GreenDroid may use to develop efficient processors for Smart phones.

5. CONCLUSION:

In this paper, we presented a study of realenergy problems in Smartphone. We have used GreenDroid tool to diagnosis the energy problems in Android phones. This tool is only used to list out the problem of smartphone applications but cannot solve the issues like energy optimization. So we are going to focus on this study.Our study results confirmed the effectiveness and practical usefulness of GreenDroid for energy optimization.

REFERENCES:

[1]A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices," in Proc. ACM Workshop Hot Topics Netw., 2011, pp. 5:1–5:6.

[2] Android Sensor Management. (2013). [Online].Available:http://developer.android.com/reference/android/hardware/SensorManager.html

[3] Android power management. (2013). [Online].Available:http://developer.android.com/reference/android/os/PowerManager.html.

[4] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng., 2012, pp. 59:1–59:11.

[5]"Android Process Lifecycle" (2013). URL: http://developer.android.com/reference/android/os/PowerManger.html

[6] "Android Power Management." URL: http://develpers.android.com/reference/android/os/PowerManger.html.

[7] "Android Activity Lifecycles." URL: http://developer.android.com/guide/components/activities.html.

[8]W. Visser, K. Havelund, G. Brat, and S. Park, "Model checking programs," in Proc. Int. Conf. Automated Softw. Eng., 2000, pp. 3–11.

[9] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in Proc. 35th Int. Conf. Softw. Eng., pp. 92–101.

[10] C. S. Jensen, M. R. Prasad, and A. Møller, "Automated testing with targeted event sequence generation," in Proc. Int. Symp.Softw. Testing Anal., 2013, pp. 67–77.