Journal of Network and Computer Applications 36 (2013) 1208–1218

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

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

Booting, browsing and streaming time profiling, and bottleneck analysis on android-based systems

Ying-Dar Lin [a], Cheng-Yuan Ho [b], Yuan-Cheng Lai [c,*], Tzu-Hsiung Du [a], Shun-Lee Chang [a]

[a] Department of Computer Science, National Chiao Tung University, Hsinchu County 300, Taiwan
[b] Information and Communications Technology Lab, Microelectronics and Information Systems Research Center, National Chiao Tung University, Hsinchu County 300, Taiwan
[c] Department of Information Management, National Taiwan University of Science and Technology, Taipei County 106, Taiwan

ARTICLE INFO

Article history:
Received 17 July 2012
Received in revised form
27 January 2013
Accepted 17 February 2013
Available online 4 March 2013

Keywords:
Android
Booting
Browsing
Streaming
Time profiling

ABSTRACT

Android-based systems perform slowly in three scenarios: booting, browsing, and streaming. Time profiling on Android devices involves three unique constraints: (1) the execution flow of a scenario invokes multiple software layers, (2) these software layers are implemented in different programming languages, and (3) log space is limited. To compensate for the first and second constraints, we assumed a staged approach using different profiling tools applied to different layers and programming languages. As for the last constraint and to avoid generating enormous quantities of irrelevant log data, we began profiling scenarios from an individual module, and then iteratively profiled an increased number of modules and layers, and finally consolidated the logs from different layers to identify bottlenecks. Because of this iteration, we called this approach a staged iterative instrumentation approach. To analyze the time required to boot the devices, we conducted experiments using off-the-shelf Android products. We determined that 72% of the booting time was spent initializing the user-space environment, with 44.4% and 39.2% required to start Android services and managers, and preload Java classes and resources, respectively. Results from analyzing browsing performance indicate that networking is the most significant factor, accounting for at least 90% of the delay in browsing. With regard to online streaming, networking and decoding technologies are two most important factors occupying 77% of the time required to prepare a 22 MB video file over a Wi-Fi connection. Furthermore, the overhead of this approach is low. For example, the overhead of CPU loading is about 5% in the browsing scenario. We believe that this proposed approach to time profiling represents a major step in the optimization and future development of Android-based devices.

1. Introduction

Devices such as Smartphones, set-top boxes, and netbooks provide users with the ability to access the Internet at anytime, from anywhere. Among these Internet connectable devices, Smartphones operating under Android, an open source platform developed from Linux in 2009, are expected to garner the most attention in coming years. Designing a device on the Android operating system reduces licensing fees, and developers benefit from the ability to develop new features and test their innovations in an open source environment (Pieterse and Olivier, 2012). However, Android-based devices suffer from poor performance in three areas: booting, browsing, and streaming. Boot-up time is the first perception users have when trying a new Smartphone, and browsing and streaming are two usage scenarios commonly encountered by Smartphone subscribers (Comscore.com, 2008). We compared the time spent booting and browsing using popular off-the-shelf products. Although all of these products have similar hardware capabilities, the execution on Android-based products takes much longer than similar applications on iPhones. Previous researchers have worked intensively on improving the performance in these three areas (Singh et al., 2011; Zhao et al., 2011; Trestian et al., 2012), and all of these studies have shared three common procedures. First, profiling tools were used to trace the flow of execution and the running time of targeted tasks. Second, the flow of execution was redesigned to reduce the time required to perform the three tasks. Finally, the improvement in performance was evaluated by profiling the system again. Clearly, profiling tools play an important role in the enhancement of performance. Profiling tools can be categorized into two types: instrumentation and sampling techniques (Ghoroghi and Alinaghi, http://www.docstoc.com/docs/7671023/An-introduction-to-profiling-mechanisms-and-Linux-profilers; Patel and Rajawat, 2011). Instrumentation techniques, such as debug classes (Yoon, 2012;

* Corresponding author. Tel.: +886 2 27376794; fax: +886 2 27376777.
E-mail address: laiyc@cs.ntust.edu.tw (Y.-C. Lai).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.02.024

Android Developer, http://developer.android.com/reference/packages.html), log utilities (Yoon, 2012; Android Developer, http://developer.android.com/reference/packages.html), printk (Printk Times, http://elinux.org/Printk_Times), Linux Trace Toolkit Next Generation (LTTng) (Toupin, 2011), and Kernel Function Trace (KFT) (http://elinux.org/Kernel_Function_Trace) insert profiling code into the source code of targeted programs, thereby enabling the profiling results to be collected during execution. On the other hand, sampling techniques, such as OProfile (Levon, http://oprofile.sourceforge.net/) and Bootchart (Mahkovec, http://www.bootchart.org/), collect the process-usage statistics by periodically checking which program or process is occupying the CPU.

Two difficulties have consistently plagued previous studies. One problem is the fact that Android is a complex system with multiple layers, and the characteristics of each profiling tool limit it to a specific layer or layers within the software, as explained in Section 2. The other problem is that previous researchers have validated their ideas on development boards or emulators, despite the fact that hardware is meant to be optimized for commercial products. This has resulted in discrepancies between the performance results obtained in the lab and those provided by off-the-shelf products. Unlike a development board equipped with sufficient log space, hardware optimization may leave only limited space on the end-product, e.g., a 64 KB log buffer on the HTC Dream Smartphone. As a result, profiling tools work very effectively on development boards but often encounter out-of-resource problems on the devices for which they were intended.

This work proposes a novel approach to profiling across multiple layers to identify true bottlenecks in booting, browsing, and streaming using real-world Android based devices. We revealed common profiling procedures used for arbitrary scenarios and developed specific profiling procedures for each scenario. All procedures were validated on off-the-shelf products, to identify the true bottlenecks of each scenario. This work has the following major contributions: (1) we propose a staged iterative instrumentation approach, which has properties of limited log space, multi-layers, and multi-programming-languages; (2) we solve the implementation issues of this approach for time profiling booting, browsing, and streaming using real-world Android devices; and (3) we conduct extensive evaluations in booting, browsing, and streaming scenarios and identify the bottlenecks in these scenarios.

The remainder of this paper is organized as follows. In Section 2, we briefly describe the Android architecture and various profiling tools. In Sections 3 and 4, we present our proposed methodologies and the means by which the profiling procedures are implemented. In Section 5, we present the experimental environment and discuss the profiling results. Finally, in Section 6, we off conclusions and suggest directions for future research.

2. Background

This section briefly describes the Android architecture and various profiling tools.

2.1. Android architecture

As shown in Fig. 1, Android software comprises four major layers, written in three different programming languages: Java, C++, and C. From basic hardware compliance to the level controlled by users, the four software layers are the Linux kernel, running environment, application framework, and applications.

1. Linux kernel layer
   The Android kernel was derived from the Linux 2.6 kernel, so it inherits many advantages of Linux such as numerous device drivers and core operating system functionalities, e.g., memory management, process management, and networking. In order to accommodate the tightly constrained resources associated with embedded devices, Android adds new modules into its kernel or modifies some parts of the Linux kernel. For example, Android includes Yet Another Flash File System, 2nd edition (Yaffs2), an optimized file system for NAND flash, but the Linux kernel did not. Accordingly, Android can run on many different types of devices. Moreover, most of the profiling tools used in common Linux distributions can be adopted for Android since the entire Linux kernel is written in C programming language.
2. Running environment layer
   The running environment layer includes two major components, native libraries, and Android runtime. Native libraries contain a set of C and C++ libraries, such as libc and OpenGL/ES, providing common routines for upper layers. In contrast, Android runtime is designed specifically for Android to meet the needs of operating in a resource-limited embedded device. It includes Dalvik virtual machine (VM) and core libraries. Dalvik VM is derived from Java VM and written in C, C++, and Java while the core libraries are written in Java containing common Java classes for the development of applications.
3. Application framework layer
   The application framework layer contains reusable components, accessible to applications. Components in this layer are written in Java, C++, and C programming languages. Among the components in this layer, activity and window managers are the two most important. The former manages the life cycle of applications, while the latter draws graphic elements, such as status bars, to provide a foreground graphical-user-interface (GUI). Furthermore, in Android, only one foreground GUI application may be displayed at a given time, while other running applications are managed by the activity manger in the background.
4. Applications layer
   All user-visible applications on Android can be developed by anyone, like commercial developers, open-source communities, and Google, and are written in Java programming language.

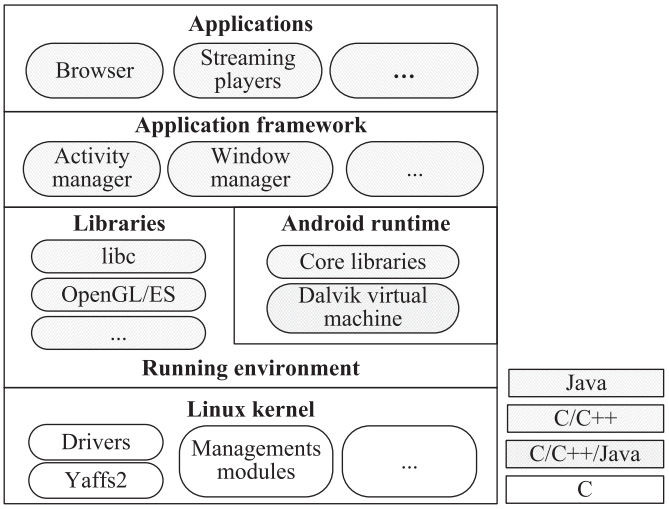

Fig. 1. Android architecture.

2.2. Profiling tools on android

Profiling tools are used to detect hotspots in a program or a set of programs to alleviate performance issues. A hotspot is a piece of code that is frequently executed or the execution time of which