Empirical assessment of the real-time capabilities of Android

CSE-605 Checkpoint 1 - Rex

Xia Wu
xiawu@buffalo.edu

Xingxiao Yuan
xingxiao@buffalo.edu

Michael Wehar
mwehar@buffalo.edu

Yihong Chen
ychen78@buffalo.edu

March 12, 2015

Contents

1 Introduction **1**

2 Android Components **2**

3 Intents/Receivers **3**
- 3.1 Single Process . . . 3
- 3.2 Multiple Processes . . . 3
  - 3.2.1 Garbage Collector . . . 3
  - 3.2.2 Synchronization . . . 4
  - 3.2.3 Scheduler . . . 4

4 TODO Content Providers **4**

5 TODO Processes/Threads **4**

6 TODO AlarmManager **4**

7 DONE Parcelable/Serializable **4**

1 Introduction

Android is a mobile operating system based on the Linux kernel and is developed by Google. Along with the popularity of smartphones, 3G devices, and other advanced mobile applications around 2009, Android has been standing out in the so-called "smart wars" with its counterpart such as Apple iOS and Windows, for its being a ready-made, low-cost, and customizable operating system (OS). According to Strategy Analytics' latest report (2014), Android is getting dangerously close to worldwide domination with a record 80% of all smartphone running the mobile OS. What's more, Android has gone a step further – it's the main driver behind the continuing climb of the smartphone industry; Android now accounts for an impressive 8 in 10 of all smartphones shipped on the planet.

Enjoying such a big fat market share, Android facilitates its rapid deployment in many domains. Android was originally designed to be used in mobile computing applications, from handsets to tablets to e-books. But developers are also looking to employ Android in a variety of other embedded systems that have traditionally relied on the benefits of true real-time operating systems. What makes an operating system real-time is that on the top of functioning correctly, its ability to meet deadlines. Time constraints for system performance are critical to such real-time embedded systems that mandate response to stimuli within pre-specified real-time design specifications. Reliability consideration of utilizing a system as real time OS requires a detailed evaluation of the ability of this system to meet these specifications, in other words, to be predictable.

Android enables developers to write applications primarily in Java. Because of its managed memory system, traditional Java Virtual Machine (JVM) always raises concerns when it comes to real-time applications, how well does Android's Process Virtual Machine Dalvik act in the real-time domain? This is what our project Empirical assessment of the real-time capabilities of Android is going to find out! While majority of studies on the reliability of Android system focus upon the functional failures of activities; our research sets emphasis on studying the time-related behavior of Android for memory exhausting activities, analyzing the possible intrinsic obstacle(s) of Android system for real-time applications. More specifically, we design/apply benchmarks suitable for Android construct – targeting at the Android DragonBoard™ 800 development board that is based on the Qualcomm® Snapdragon™ 800 processor (APQ8074), observe Android garbage collector's performances, and test Android program's predictability. At the end of the process, we hope to come up with a set of improvement suggestion to help Android more real-time friendly.

2 Android Components

- Intents and Intent Filters (receivers)

  We will evaluate intent delivery mechanism of Android. The intents will appear for almost all multiple processes experiments.

- Activities and App Widgets

  We will not directly evaluate activities and app widgets because they're only related with UI, it's hard to produce convincible result because there're too many elements out of control, like the GPU power, the screen resolution.

- Services

  We will not directly evalute services as well. The reason is same as activities.

- Content Providers

  We will evalute content providers related **Garbage Collector**, **Synchronization**, and **Scheduler**. Because it's shared data mechanism Android provied.

- Processes and Threads

  We will evaluate processes and threads directly. They're both our targets and mechanisms to use.

3 Intents/Receivers

3.1 Single Process

To directly evaluate intents and receivers on single process deosn't make sense. But this can provide **baseline for our experiments**. We can compare the results of same experiment on single process and multiple processes. Then we can get reasoning of different behaviour.

3.2 Multiple Processes

We can use process(es) to generate bunch of intents, then use other process(es) to receive the intents. In addition, there're always background process(es) with some computation to produce pressure for Android system. So we can evaluate the order and time of intent delivery. It can provide some pressure for **Garbage Collector**, **Synchronization**, and **Scheduler**.

3.2.1 Garbage Collector

We can use one sender and one receiver to test Garbage Collector, So we can evaluate how garbage collector works:

- frequency of grabage collection and memory pressure
- running time of grabage collection and memory pressure

The memory pressure should contains different types:

	big objects	medium objects	small objects
long live time	X	X	X
short live time	X	X	X

So where pressure come from? To evalute the behavior of Android system, it need some pressure for different components. So that we can infer the predictability of different conponents, and the interaction between different components. The pressure can come from:

- Other background process(es) with computation
- Computation inside senders
- Computation inside receivers

We will divide our experiments into three phases:

Phase 1. we only have pressure from background process(es). It's easier to implement and tune for different memory pressure types.

Phase 2. we'll add additional computation to senders and receivers to compare whether computation source affect Android's performance.

Phase 3. we combined the different pressure together to get final evaluation.

The computation can be the task from SPECjvm2008 (Java Virtual Machine Benchmark) or DaCapo Benchmark. In addition, we can assoicate the Parcelable vs. Serializable with experiment phase 3.

3.2.2 Synchronization

This task need other Android components. Because we can not pass an object as extra of an intent, we need serialize the object first. So there no directly synchronized mechanism between sender and receiver, but we can pass some metadata to let receivers use something need synchronization like **Content Provider**. We'll discuss in next section.

3.2.3 Scheduler

We can use multiple background processes to provide pressure for scheduler. Then we use the order of intent delivery to evaluate scheduler and intent delivery mechanism. More details can be found in Section Processes/Threads.

4 TODO Content Providers

Need focus on synchronization

5 TODO Processes/Threads

6 TODO AlarmManager

It's a critical factor for real-time system.

7 DONE Parcelable/Serializable

According to this blog, parcelable mechanism have 10 times better performance than serializable mechanism. But parcelable need developers to implement writeToParcel and createFromParcel manually. So parcelable can save the overhead to iterate all fields of object. But we can compare the two mechanisms by how much pressure they generate to garbage collector.

The approach is to pass same amount of objects from one process to another process (either the same process or alien), then we compare the different behaviors of garbage collector. It's possible to evaluate scheduler as well.

In conclude, the parcelable and serializable mechanisms are methods to provide pressure for Android system. In the meantime, we can evalute the performance of them. The result may improve static code analysis of Andorid codes.