Instructor: Li Erran Li
(lierranli@cs.columbia.edu)

http://www.cs.columbia.edu/~coms6998-8/

3/19/2012: Smart phone virtualization and storage
Announcements

• Preliminary project report due next week March 26\textsuperscript{th}
• There will be two advanced programming lab sessions: one for iOS and one for Android
  – Email me the topics you would like to cover
Smart Phone Virtualization

Cells video demo
Virtualization
Server Virtualization

Bare-Metal Hypervisor

Hypervisor / VMM

Hardware

OS Kernel

OS Kernel

OS Kernel

Courtesy: Jason Nieh et al.
Desktop Virtualization

Hosted Hypervisor

Host OS Kernel

OS

OS

OS

Hypervisor / VMM

kernel module

emulated devices

Hardware

poor device performance

Courtesy: Jason Nieh et al.
Non-Virtualization

User Space SDK

- no standard apps
- less secure

- custom user space API for isolated apps

OS Kernel

Hardware

Cellular Networks and Mobile Computing
(COMS 6998-8)

Courtesy: Jason Nieh et al.
Key Challenges

• device diversity

<table>
<thead>
<tr>
<th>Power</th>
<th>Touchscreen</th>
<th>Buttons</th>
<th>WiFi</th>
<th>GPU</th>
<th>Framebuffer</th>
<th>GPS</th>
<th>Compass</th>
<th>pmem</th>
<th>Binder IPC</th>
<th>Accelerometer</th>
<th>Accelerated UI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Radio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h.264 accel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>camera(s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• mobile usage model

⇒ graphics

3/19/12

Cellular Networks and Mobile Computing
(COMS 6998-8)

Courtesy: Jason Nieh et al.
Cells

Key Observation

large: lots of windows/apps

one app at a time

Courtesy: Jason Nieh et al.
Cells
Key Observation

screen real-estate is limited, and mobile phone users are accustomed to interacting with *one thing* at time
Cells

Usage Model

foreground / background
Cells
Complete Virtualization

• multiple, isolated virtual phones (VPs) on a single mobile device
• 100% device support in each VP
  ▸ unique phone numbers - single SIM!
  ▸ accelerated 3D graphics!
Cells

Efficient Virtualization

- less than 2% overhead in runtime tests
- imperceptible switch time among VPs
Single Kernel: Multiple VPs

isolated collection of processes

virtualize at OS interface

Linux Kernel

Courtesy: Jason Nieh et al.
Single Kernel: Device Support

- Power
- Cell Radio
- hw codec
- camera(s)
- Touchscreen
- WiFi
- pmem
- speakers
- microphone
- Buttons
- GPU
- Binder IPC
- Accelerometer
- headset
- GPS
- Framebuffer
- Compass
- RTC / Alarms
Single Kernel: Device Support

all VPs access the same device simultaneously
Device Namespaces

safely, correctly multiplex access to devices

Linux Kernel

device namespaces

WiFi
Cell Radio
Framebuffer
GPU
Power
Input
Sensors
Audio/Video
RTC / Alarms
... Android...

VP 1
VP 2
VP 3

...
Cells

device namespaces

foreground / background

Complete, Efficient, Transparent Mobile Virtualization
efficient basic graphics virtualization

hardware accelerated graphics

proprietary/closed interface

Linux Kernel

device namespaces

WiFi
Cell Radio
Framebuffer
GPU
Power
Input
Sensors
Audio/Video
RTC / Alarms
Android...
Approach 1: Single Assignment

virtual addresses
physical addresses

screen memory

Framebuffer
Approach 2: Emulated Hardware

Framebuffer

virtual state

screen memory

emulated framebuffer

virtual state

Framebuffer

Courtesy: Jason Nieh et al.
Cells: Device Namespaces

mux_fb presents identical device interface to all VPs using device namespaces

swap virt addr mappings point to different phys addr

screen memory

Courtesy: Jason Nieh et al.
Accelerated Graphics

VP: just a set of processes!

OpenGL context

Framebuffer

GPU

MMU

Process isolation

graphics virtual addresses

physical addresses

 Courtesy: Jason Nieh et al.
Device Namespace + Graphics Context

- VP 1 (foreground)
  - OpenGL context
- VP 2 (background)
  - OpenGL context
- VP 3 (background)
  - OpenGL context

Screen memory

Graphics virtual addresses

Physical addresses

GPU

MMU
VoIP?

![Diagram showing the relationship between RiLD, Vendor RIL, VoIP, Linux Kernel, Drivers, Baseband: GSM / CDMA, and RiID.]

Courtesy: Jason Nieh et al.
Dual-SIM?

Cellular Networks and Mobile Computing
(COMS 6998-8)

3/19/12

Courtesy: Jason Nieh et al.
Cells: User-Level Namespace Proxy

proprietary hardware/software requires a well-defined interface.
Experimental Results

Setup

- Nexus S
- five virtual phones
- overhead vs. stock (Android 2.3)
Experimental Results

Setup

• CPU ⟨Linpack⟩
• graphics ⟨Neocore⟩
• storage ⟨Quadrant⟩
• web browsing ⟨Sun Spider⟩
• networking ⟨Custom WiFi Test⟩
Experimental Results
Runtime Overhead

Negligible Overhead In 3D Measurements!

Courtesy: Jason Nieh et al.
Cells

Complete, Efficient, Transparent Mobile Virtualization

- device namespaces
  - safely and efficiently share devices
- foreground / background
  - designed specifically for mobile devices
- implemented on Android
- less than 2% overhead on Nexus S
More Info

cells.cs.columbia.edu

cellrox.com

Courtesy: Jason Nieh et al.
Revisiting Storage for Smartphones

Hyojun Kim  Nitin Agrawal  Cristian Ungureanu
Background

• **blktrace**: collect block level traces for device I/O

• **monkeyrunner**
  – installed at `android-sdk-macosx/tools/monkeyrunner`
  – functional testing framework for Interactive Android applications
blktrace

• Block IO layer

Source: Alan D. Brunell
http://www.gelato.org/pdf/apr2006/gelato_ICE06apr_blktrace_brunelle_hp.pdf

Cellular Networks and Mobile Computing (COMS 6998-8)
blktrace (Cont’d)

**blktrace**: General Architecture

Source: Alan D. Brunell
http://www.gelato.org/pdf/apr2006/gelato_ICE06apr_blktrace_brunelle_hp.pdf
blktrace (Cont’d)

• blktrace sample traces

```
% blktrace -d /dev/sda -o - | blkpars...[kjournald]
 8,0 3 1 0.0000000000 697 G W 223490 + 8 [kjournald]
 8,0 3 2 0.000001829 697 P R [kjournald]
 8,0 3 3 0.000002197 697 Q W 223490 + 8 [kjournald]
 8,0 3 4 0.000005533 697 M W 223498 + 8 [kjournald]
 8,0 3 5 0.000008607 697 M W 223506 + 8 [kjournald]
...
 8,0 3 10 0.000024062 697 D W 223490 + 56 [kjournald]
 8,0 1 11 0.009507758 0 C W 223490 + 56 [0]
```

Source: Alan D. Brunell
http://www.gelato.org/pdf/apr2006/gelato_ICE06apr_blktrace_brunelle_hp.pdf
Example code

1. # Imports the monkeyrunner modules used by this program
2. from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice
3. def main():
4.     # Connects to the current device, returning a MonkeyDevice object
5.     device = MonkeyRunner.waitForConnection()
6.     print 'waiting for connection...
'
7.     package = 'coms6998.cs.columbia.edu'
8.     activity = 'coms6998.cs.columbia.edu.VoiceRecognitionDemoActivity'
9.     # sets the name of the component to start
10.    runComponent = package + '/' + activity
11.    # Runs the component
12.    device.startActivity(component=runComponent)
13.    # Presses the speaker button
14.    device.press('DPAD_DOWN', MonkeyDevice.DOWN_AND_UP)
15.    device.press('DPAD_CENTER', MonkeyDevice.DOWN_AND_UP)
16.    # Takes a screenshot
17.    screenshot = device.takeSnapshot()
18.    # Writes the screenshot to a file
19.    screenshot.writeToFile('./device1.png', 'png')
20.    reference = MonkeyRunner loadImageFromFile('./device.png')
21.    if not screenshot.sameAs('./device.png', 0.9):
22.        print "comparison failed!
"
23. if __name__ == '__main__':
24.    main()
monkeyrunner

• Demo
Life in the “Post-PC” Mobile Era

• Smartphone and tablet markets are huge & growing
  – 100 Million smartphones shipped in Q4 2010, 92 M PCs [IDC]
  – Out of 750 Million Facebook users, 250 Million (& growing) access through mobile; mobile users twice as active [FB]

• Innovation in mobile hardware: packing *everything* you need in your pocket
  – Blurring the phone/tablet divide: Samsung Galaxy Note
  – Hardware add-ons: NEC Medias (6.7mm thick, waterproof shell, TV tuner, NFC, HD camera, ..)

• Manufacturers making it easier to replace PCs
  – Motorola Atrix dock converts a phone into laptop
Waiting is undesirable!
Annoying for the user
More so for interactive mobile users
More time, more battery
Easy to lose customers

Aren’t network and CPU the real problem?
Why are we talking about storage?
Understanding Mobile Performance

Well understood!

• Network performance can impact user experience
  – 3G often considered the bottleneck for apps like browsing
  – Service providers heavily investing in 4G and beyond

CPU and graphics performance crucial as well
  – Plenty of gaming, video, flash-player apps hungry for compute

Not well understood!

• Does storage performance impact mobile experience?
  – For storage, vendors & consumers mostly refer to capacity
Flash storage on mobile performs better than wireless networks.

Most apps are interactive; as long as performance exceeds that of the network, difficult for storage to be bottleneck.
Outline

✓ Introduction

Why storage is a problem

Android storage background and setup

Experimental results

Solutions
Why Storage is a Problem
Random versus Sequential Disparity

- Performance for random I/O significantly worse than seq; inherent with flash storage
- Mobile flash storage classified into *speed classes* based on sequential throughput
  - Random write performance is orders of magnitude worse

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Speed Class</th>
<th>Cost US $</th>
<th>Seq Write</th>
<th>Rand Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcend</td>
<td>2</td>
<td>26</td>
<td>4.2</td>
<td>1.18</td>
</tr>
<tr>
<td>RIData</td>
<td>2</td>
<td>27</td>
<td>7.9</td>
<td>0.02</td>
</tr>
<tr>
<td>Sandisk</td>
<td>4</td>
<td>23</td>
<td>5.5</td>
<td>0.70</td>
</tr>
<tr>
<td>Kingston</td>
<td>4</td>
<td>25</td>
<td>4.9</td>
<td>0.01</td>
</tr>
<tr>
<td>Wintec</td>
<td>6</td>
<td>25</td>
<td>15.0</td>
<td>0.01</td>
</tr>
<tr>
<td>A-Data</td>
<td>6</td>
<td>30</td>
<td>10.8</td>
<td>0.01</td>
</tr>
<tr>
<td>Patriot</td>
<td>10</td>
<td>29</td>
<td>10.5</td>
<td>0.01</td>
</tr>
<tr>
<td>PNY</td>
<td>10</td>
<td>29</td>
<td>15.3</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Performance MB/s

However, we find that for several popular apps, substantial fraction of I/O is random writes (including web browsing!)
• Storage coming under increasingly more scrutiny in mobile usage
  – Random I/O performance has not kept pace with network improvements
  – 802.11n (600 Mbps peak) and 802.11ad (7 Gbps peak) offer potential for significantly faster network connectivity to mobile devices in the future
Deconstructing Mobile App Performance

• **Focus:** understanding contribution of storage
  – How does storage subsystem impact performance of popular and common applications on mobile devices?
  – Performed analysis on Android for several popular apps

• Several interesting observations through measurements
  – Storage adversely affects performance of even interactive apps, including ones not thought of as storage I/O intensive
  – SD Speed Class not necessarily indicative of app performance
  – Higher total CPU consumption for same activity when using slower storage; points to potential problems with OS or apps

• Improving storage stack to improve mobile experience
Outline

✓ Introduction

✓ Why storage is a problem
  Android storage background and setup

Experimental results

Solutions
# Storage Partitions on Android

<table>
<thead>
<tr>
<th>Partition</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misc</td>
<td>H/W settings, persistent shared space between OS &amp; bootloader</td>
</tr>
<tr>
<td>Recovery</td>
<td>Alternative boot-into-recovery partition for advanced recovery</td>
</tr>
<tr>
<td>Boot</td>
<td>Enables the phone to boot, includes the bootloader and kernel</td>
</tr>
<tr>
<td>System</td>
<td>Contains the remaining OS, pre-installed system apps; read-only</td>
</tr>
<tr>
<td>Cache</td>
<td>Used to stage and apply “over the air” updates; holds system images</td>
</tr>
<tr>
<td>Data</td>
<td>Stores user data (e.g., contacts, messages, settings) and installed apps; SQLite DB containing app data also stored here. Wiped on factory reset</td>
</tr>
<tr>
<td>Sdcard</td>
<td>External SD card partition to store media, documents, backup files etc</td>
</tr>
<tr>
<td>Sd-ext</td>
<td>Non-standard partition on SD card that can act as data partition</td>
</tr>
</tbody>
</table>

Internal NAND Flash Memory (512MB)

- `/misc` 896KB
  - rootfs
  - 4MB alternate boot

- `/recovery` rootfs
  - 3.5MB kernel

- `/boot` yaffs2
  - 145MB read-only

- `/system` yaffs2
  - 95MB read write

- `/cache` yaffs2
  - 196.3MB read write

- `/data` yaffs2
  - 196.3MB read write

- `/sdcard` FAT32
  - 16GB read write

---

3/19/12

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Nitin Agrawal et al.
Phone and Generic Experimental Setup

• Rooted and set up a Google Nexus One phone for development
  – GSM phone with a 1 GHz Qualcomm QSD8250 Snapdragon processor
  – 512 MB RAM, and 512 MB internal flash storage

• Setup dedicated wireless access point
  – 802.11 b/g on a laptop for WiFi experiments

• Installed AOSP (Android Open Source Project)
  – Linux kernel 2.6.35.7 modified to provide resource usage information
Custom Experimental Setup

Requirements beyond stock Android

• Ability to compare app performance on different storage devices
  – Several apps heavily use the internal non-removable storage
  – To observe and measure all I/O activity, we modified Android’s init process to mount all internal partitions on SD card
  – Measurement study over the internal flash memory and 8 external SD cards, chosen 2 each from the different SD speed classes

• Observe effects of shifting bottlenecks w/ faster wireless networks
  – But, faster wireless networks not available on the phones of today
  – Reverse Tethering to emulate faster networks: lets the smartphone access the host computer’s internet connection through a wired link (miniUSB cable)

• Instrumentation to measure CPU, storage, memory, n/w utilization

• Setup not typical but allows running what-if scenarios with storage devices and networks of different performance characteristics
Apps and Experiments Performed

**WebBench Browser**
Visits 50 websites
Based on WebKit
Using HTTP proxy server

**App Install**
Top 10 apps on Market

**App Launch**
Games, Weather, YouTube
GasBuddy, Gmail, Twitter,
Books, Gallery, IMDB

**RLBench SQLite**
Synthetic SQL benchmark

**Facebook**

**Android Email**

**Google Maps**

**Pulse News Reader**

**Background**
**Apps:** Twitter, Books, Gmail
Contacts, Picasa, Calendar
**Widgets:** Pulse, YouTube,
News, Weather, Calendar,
Facebook, Market, Twitter

3/19/12

Cellular Networks and Mobile Computing
(COMS 6998-8)

Courtesy: Nitin Agrawal et al.
Outline

✓ Introduction

✓ Why storage is a problem

✓ Android storage background and setup

**Experimental results (talk focuses on runtime of apps)**

Paper has results on I/O activity, CPU, App Launch behavior, etc

Solutions
Runtime on WiFi varies by 2000% between internal and Kingston
  - Even with repeated experiments, with new cards across speed classes
Even without considering Kingston, significant performance variation (~200%)
Storage significantly affects app performance and consequently user experience
With a faster network (USB in RT), variance was 222% (without Kingston)

**With 10X increase in N/W speed, hardly any difference in runtime**
Runtime on WiFi varies by 2000% between internal and Kingston
- Even with repeated experiments, with new cards across speed classes
Even without considering Kingston, significant performance variation (~200%)
Storage significantly affects app performance and consequently user experience
With a faster network (USB in RT), variance was 222% (without Kingston)

**With 10X increase in N/W speed, hardly any difference in runtime**
We find a similar trend for several popular apps. Storage device performance important, better card $\rightarrow$ faster apps.

Apart from the benefits provided by selecting a good flash device, are there additional opportunities for improvement in storage?
WebBench: Sequential versus Random I/O

- Few reads, mostly at the start; significantly more writes
- About 2X more sequential writes than random writes
- Since rand is worse than seq by >> 2X, random dominates
- Apps write enough randomly to cause severe performance drop

Paper has a table on I/O activity for other apps

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Seq:Rand perf ratio</th>
<th>Rand IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transcend</td>
<td>4</td>
<td>302</td>
</tr>
<tr>
<td>Sandisk</td>
<td>8</td>
<td>179</td>
</tr>
<tr>
<td>RiData</td>
<td>395</td>
<td>5</td>
</tr>
<tr>
<td>Kingston</td>
<td>490</td>
<td>2.6</td>
</tr>
<tr>
<td>Wintec</td>
<td>1500</td>
<td>2.6</td>
</tr>
<tr>
<td>A-Data</td>
<td>1080</td>
<td>2.6</td>
</tr>
<tr>
<td>Patriot</td>
<td>1050</td>
<td>2.6</td>
</tr>
<tr>
<td>PNY</td>
<td>1530</td>
<td>2.6</td>
</tr>
</tbody>
</table>

3/19/12

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Nitin Agrawal et al.
How Apps Use Storage?

- Exactly what makes web browsing slow on Android?
  - Key lies in understanding how apps use SQLite and FS interface

  ![WebBench Storage Schema Diagram]

  - Apps typically store some data in FS (e.g., cache files) and some in a SQLite database (e.g., cache map)
  - All data through SQLite is written synchronously \(\rightarrow\) slow!
  - Apps often use SQLite oblivious to performance effects

3/19/12

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Nitin Agrawal et al.
What-If Analysis for Solutions

What is the potential for improvements?
– E.g., if all data could be kept in RAM?
– Analysis to answer hypothetical questions

A. Web Cache in RAM
B. DB (SQLite) in RAM
C. All in RAM
D. All on SD w/ no-sync

Placing Cache on Ramdisk does not improve perf. much
DB on Ramdisk alone improves perf. significantly
Both Cache and DB on SD without sync recoups most perf
Both Cache and DB in RAM → no extra benefit

Both in RAM → no extra benefit

A. Web Cache in RAM
B. DB (SQLite) in RAM
C. All in RAM
D. All on SD w/ no-sync

WebBench on RiData
Implications of Experimental Analysis

• Storage stack affects mobile application performance
  – Depends on random v/s sequential I/O performance
• Key bottleneck is "wimpy" storage on mobile devices
  – Performance can be much worse than laptops, desktops
  – Storage on mobile being used for desktop-like workloads
• Android exacerbates poor storage performance through synchronous SQLite interface
  – Apps use SQLite for functionality, not always needing reliability
  – SQLite write traffic is quite random → further slowdown!
• Apps use Android interfaces oblivious to performance
  – Browser writes cache map to SQLite; slows cache writes a lot
Outline

✓ Introduction

✓ Why storage is a problem

✓ Android storage background and setup

✓ Experimental results

Solutions
Pilot Solutions

- RAID-0 over SD card and internal flash
  - Leverage I/O parallelism already existent
  - Simple software RAID driver with striped I/O
  - As expected speedup, along with super linear speedup due to flash idiosyncrasies (in paper)

- Back to log-structured file systems
  - Using NilFS2 to store SQLite databases
  - Moderate benefit; suboptimal implementation

- Application-specific selective sync
  - Turn off sync for files that are deemed async per our analysis (e.g., WebCache Map DB)
  - Benefits depend on app semantics & structure

- PCM write buffer for flash cards
  - Store performance sensitive I/O (SQLite DB)
  - Small amount of PCM goes a long way
Conclusion

• Contrary to conventional wisdom, storage does affect mobile application performance
  – Effects are pronounced for a variety of interactive apps!

• Pilot solutions hint at performance improvements
  – Small degree of application awareness leads to efficient solutions
  – Pave the way for robust, deployable solutions in the future

• Storage subsystem on mobile devices needs a fresh look
  – We have taken the first steps, plenty of exciting research ahead!
  – E.g., poor storage can consume excessive CPU; potential to improve energy consumption through better storage
Questions?