Trying to Improve Android Boot Time With Readahead

Tim Bird

Sony Network Entertainment

Agenda

- Why am I looking at this?
- What am I looking at?
- What problems did I encounter?
  - Theories vs. results
- What next?
- Resources

Why am I looking at this?

- Sony Internet TV

- Sony now does Android
- Boot time is important for consumers
- Current product has about 30 second cold boot time
- Product has an option for suspend/resume, but this consumes standby power

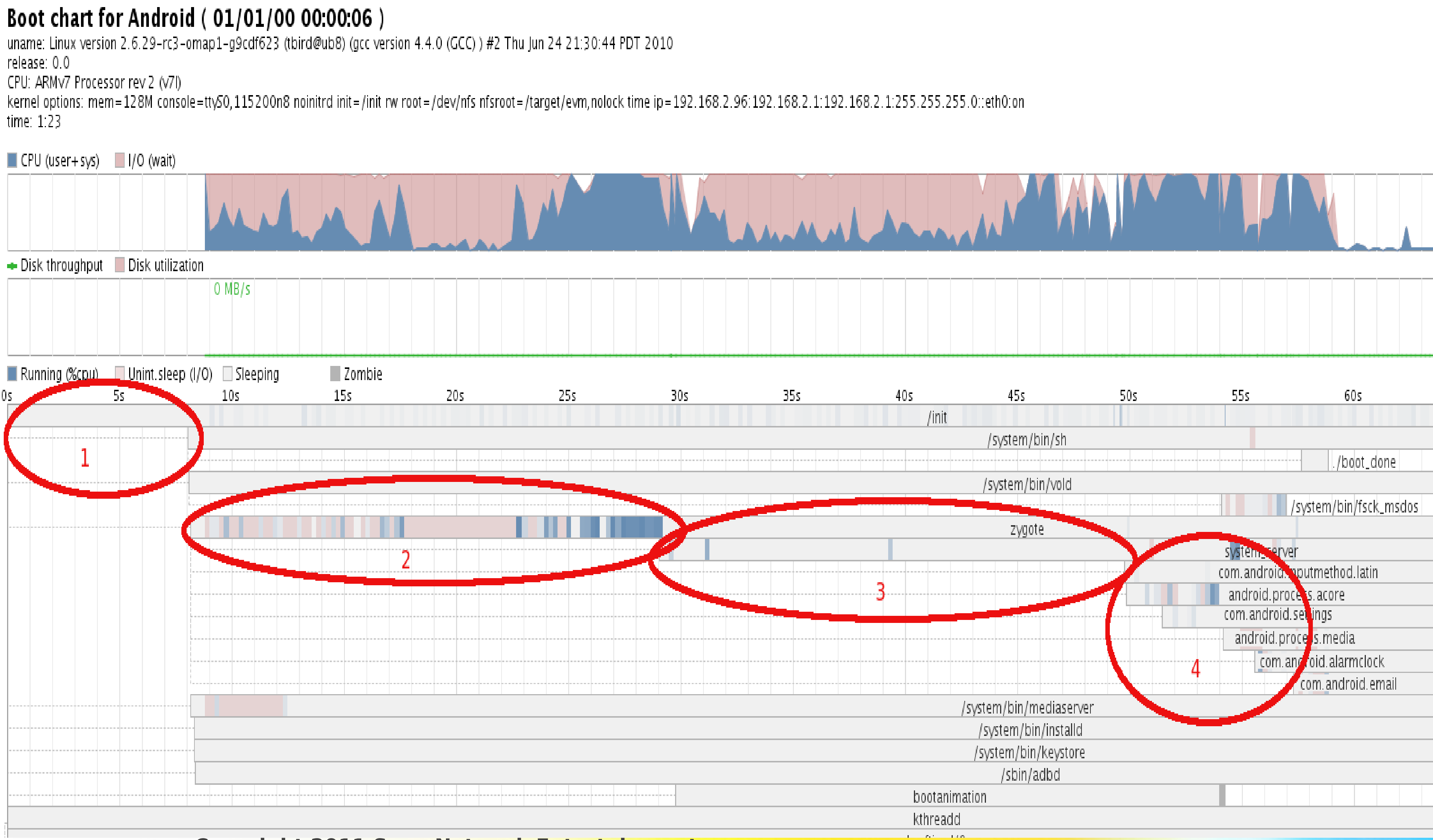
What am I looking at?

- Android boot time in general
- Previous work:
  - Presentation at LinuxCon, US, in August
    - How to measure boot time
      - Using bootchart built into Android ‘init’ program
    - I found some inefficiencies in various parts of Android
      - Mainly package scanning and file access routines in Android
- Specifically focused now on effect of readahead on boot time

Boot outline

- Bootloader
- Kernel
- Init
  - Loads several daemons and services, including zygote
  - See /init.rc and /init.<platform>.rc
- Zygote
  - Preloads classes
  - Starts package manager
- Service manager
  - Starts java services and initial applications

Bootchart diagram



Boot chart for Android (01/01/00 00:00:06)

uname: Linux version 2.6.29-rc3-omap1-g9cdf623 (tbird@ub8) (gcc version 4.4.0 (GCC)) #2 Thu Jun 24 21:30:44 PDT 2010
release: 0.0
CPU: ARMv7 Processor rev 2 (v7l)
kernel options: mem=128M console=ttyS0,115200n8 noinitrd init=/init rw root=/dev/nfs nfsroot=/target/evm,nolock time ip=192.168.2.96:192.168.2.1:192.168.2.1:255.255.255.0::eth0:on
time: 1:23

Why readahead?

- Certain operations appear to be I/O bound instead of CPU bound
  - Class preloading
  - Package scanning
- Package scanning represents about 10% of total system boot time
- Readahead should fix this
  - How much can preloading and package scanning be reduced?
  - What would the effect be on the rest of the system?

Why readahead? (cont.)

- Readahead has potential to improve boot time with no application changes
  - I’m a kernel developer – I don’t do Java
  - But also, I sometimes don’t have access to application stacks for product teams
  - Speedups requiring no user-space changes are attractive
- Result from this work could possibly help other products

Readahead theory

- Readahead = read data into Linux page cache before it is required during boot
  - Avoids I/O latency at time of request
    - Reads from page cache are extremely fast
- Reads can be scheduled while CPU is doing other work
- Note: There’s no change in total data read or total number of “useful” CPU cycles
  - Goal is to eliminate cycles waiting for I/O

Readahead theory (cont.)

- Readahead may also allow for better I/O scheduling
  - Can optimize read order when reads are not done on demand
- I/O scheduling only makes sense if there is “setup latency” for read requests

Readahead on flash media

- Flash media has no rotational latency and no seek latency
  - Flash media has practically no setup latency – only transfer latency
- So there are theoretically no gains to be made from optimizing read order
- Only gain can be from scheduling I/O during CPU waits for other activities
  - That is, there should be no gain from CPU waits during I/O activity

Theory vs. practice

- It turns out to be easy to make things worse
- Theoretically, I shouldn’t see any performance improvement
- However – I/O scheduling in Linux is not optimized for flash media
  - Maybe there are gains from eliminating Linux scheduling heuristics that are sub-optimal for flash media
- Encouragement from Paul Mundt:
  - “Sorry Tim! On the bright side there's probably an Android conference you can take this to where you can contrast the microseconds you save with readahead to the 50 second application startup time. ;-)”

Readahead in practice

- Need a method to determine what data to readahead
  - Could instrument kernel block layer
  - Could monitor reads with strace
    - Turns out that most reads during Android package scanning are via direct memory references on mmap’ed files
  - Could monitor page faults – yuk!
- I used a combination of strace (for file open requests) and mincore (for pages in page cache)

Mincore

- Mincore = system call to list the pages in the page cache for a particular file
- I wrote a ‘mincore’ program (based on ideas and program by Scott Remnant)
- Mincore shows pages that are in-cache for files specified on the command line
- Has an option to write out a block list, for direct use with treadahead

Mincore output

```
# mincore G*.apk

Cached Blocks of Gallery3D.apk: [################################          #####
################################################################################
###########          ############################################]
   172/191 pages in cache.
Cached Blocks of GlobalSearch.apk: [##########################]
   26/26 pages in cache.
Cached Blocks of GoogleApps.apk: [##############################]
   30/30 pages in cache.
Cached Blocks of GoogleCheckin.apk: [############]
   12/12 pages in cache.
Cached Blocks of GoogleFeedback.apk: [#############]
   13/13 pages in cache.
Cached Blocks of GoogleSettingsProvider.apk: [#############]
   13/13 pages in cache.
Cached Blocks of GoogleSubscribedFeedsProvider.apk: [############]
   12/12 pages in cache.
Cached Blocks of GtvStats.apk: [##]
   2/2 pages in cache.
```

Mincore –r output

- Mincore can output information as list of file portions to read
  - Format is: filename offset:length, offset2:length2, …
  - Used by treadahead program

```
# mincore -r Gallery3D.apk
Gallery3D.apk 0:131072,172032:393216,602112:180224
```

Using strace

- To gather file list, I used strace to find open files
  - In /init.rc I replaced:

```
service zygote /system/bin/app_process …
```

with

```
service zygote /system/xbin/strace -tt
  -o/data/boot.strace /system/bin/app_process …
```

- After booting, pull /data/boot.strace to host, and grep for “open” syscalls

Treadahead

- Treadahead = test readahead (or Tim’s readahead ☺)
- Performs readahead operation
- Optionally performs other setup, to measure effect of different settings
- Can measure duration, and log /proc/diskstats before and after readahead

Treadahead usage

```
Usage: treadahead [options] [<file1> [<file2> ...]]
  -f   Read list of files from /readahead_list.txt
  -f<file>   Read list of files from <file>
  -h   Show this usage help
  -v   Show verbose messages
  -w   Re-write blocks to be preloaded, in the filesystem
       This may optimize the blocks for preloading
  -i   Reduce IO scheduler priority for treadahead to 'idle' (ionice -c 3)
  -j   Reduce IO scheduler priority for treadahead to 'best effort- low'
  -n   Set noop scheduler for sda, prior to doing readahead
  -p   Reduce scheduler priority for treadahead (nice 19)
  -l   Output log message on start and end
  -e   Do early exit after forking, so calling program can continue.
       This will redirect stdout to /dev/null, so with -v most messages are lost.
  -V   Show version message
  -s   Save /proc/diskstats at start and end of operation.
       Files are saved to /cache/debug/d1.txt and /cache/debug/d2.txt.
```

System description

- Sony Internet TV
  - Actually, the Blu-Ray player
- Intel Atom core (uniprocessor)
- Kernel version: 2.6.23
- 8 Gig Sony SSD
- System partition fileystem type: ext3
- Default I/O scheduler: CFQ

SONY
make.believe

What I wanted to see

- Readahead runs in background, not affecting or delaying other processes and their I/O
- Reduction in time to perform class pre-loading and application package scanning
- Ultimately, have system boot quicker
  – With NO software changes on device

Things I tested

- Affect of backgrounding the readahead task
- Affect of process scheduler priority
- Affect of I/O scheduler choice
- Affect of setting I/O scheduling priority
- Affect of re-writing the data in load order

Process backgrounding

- Init can launch things with ‘exec’ or ‘service’
  - ‘exec’ blocks and waits for program to exit before continuing
  - ‘service’ processes are launched asynchronously
    - Not sure the order
- I needed treadahead to start before ‘service’ items were processed by init
- Treadahead can run in background, returning control to ‘init’ immediately after forking into a daemon

Process scheduler priority

- Since I want treadahead to not intefere with other processes, I put it at a lower process scheduling priority
  - Programmatically do the same thing as ‘nice 19’
- This should mean that when treadahead wakes up from I/O wait in kernel, it is put on run queue – but won’t get scheduled unless other higher priority processes get blocked

I/O Scheduler choice

- Linux has several schedulers you can select at runtime
  - CFQ – completely fair scheduler (default)
    - Has several I/O queues with priorities
  - Deadline – tries to limit maximum service time for I/Os
  - Anticipatory – tries to guess what data to read next
  - Noop – does no optimizations
- Set by writing to sysfs
  - E.g. echo noop >/sys/block/sda/queue/scheduler

Noop scheduler

- Noop scheduler is reported as having no optimizations
  - That is, no queueing or delays for re-ordering I/O requests
- This should be optimal for SSDs
  - Actual device mapping of sector offsets to internal NAND flash address is impossible to predict
  - Any delays to re-order read requests are a waste of time

SONY
make.believe

I/O scheduler priority

- CFQ scheduler allows setting the I/O priority for a process
  - In desktop linux, this is the ‘ionice’ command
- Uses the ioprio_set() system call
- Have 3 scheduling classes:
  - Realtime, best effort, idle
- See ‘man ionice’, and Documentation/block/ioprio.txt

Re-writing data in load order

- Hypothesis – Maybe if the pages are written to SSD in readahead order, it will force pages into same erase block
- This worsened the read performance by between 2 and 13%
  - Hard to describe what happened during testing
    - Performance would be stable, then suddenly shift by 5 or 10 percent
      - Re-writing the data always made it worse
  - Possibly the effect of write amplification on meta-data

Results Summary

Event	baseline	readahead in foreground	readahead in background	readahead with nice 19	readahead with I/O priority = idle
system start	0.00	0.00	0.00	0.00	0.00
readahead start	n/a	8.74	8.74	8.74	8.74
readahead end	n/a	11.29	11.29	11.29	15.09
more stuff start	8.75	11.29	8.75	8.75	8.75
package scan start	26.42	28.97	27.20	27.08	28.77
package scan end	29.44	29.75	27.98	27.86	29.57
boot done	32.69	33.00	31.17	31.20	32.82

Results Summary (cont.)

Description	baseline	readahead in foreground	readahead in background	readahead with nice 19	readahead with I/O priority = idle
readahead duration	n/a	2.55	2.55	2.55	6.35
package scan start delay	0	2.55	0.78	0.66	2.35
package scan duration	3.02	0.78	0.78	0.78	0.80
boot time improvement	0	-0.31	1.52	1.49	-0.13
boot percent improvement	0.00%	-0.95%	4.65%	4.56%	-0.40%

Results

- Backgrounding treadahead allowed init to proceed exec-ing other programs (shocker!)
- Using ‘noop’ scheduler had no effect on treadahead duration
- Adjusting process priority had no effect on treadahead’s duration
  - It appears to have gotten scheduled just as much

Results (cont.)

- Set I/O priority to idle (ionice)
  - Converts treadahead duration from 2.5 seconds 6.3 seconds
  - But adds 2.35 seconds of delay to other processes
    - Expected to see less impact on foreground processes
  - Results in an overall longer boot

Problems encountered and workarounds

- Working with opaque system (SDD) is obnoxious
  - There’s no way to determine real location and grouping of pages
- Need to clear both Linux and SDD caches
  - echo 3 >/proc/sys/vm/drop_caches
  - Copy large file, to clear SDD cache (I'm not sure this worked, there was no change in behaviour, and I expected some)
  - Also used sync, and waited for disk to settle, after re-writes

More issues encountered

- Missing library wrappers for syscalls
  - readahead() syscall not included in bionic
  - ioprio_set() syscall not included in bionic or glibc
- Readahead call returns before reads are finished
  - Have to wait for I/O queue to settle (according to /proc/diskstats)
- Need to mask other activities on system before performing benchmarks
  - That is, use Android "stop“ command
- Timestamps between logging systems are not consistent
  - Solution was logsync – program to write to multiple logs at same time
  - Would be nice to write a tool to unify the timestamp space

SONY
make.believe

Next steps

- Try tweaks to CFQ scheduling quantums
- Want to examine on SMP system
  - Tried to get some results with PandaBoard (dual-core OMAP)
  - Flaky MMC caused unreliable timings in speed tests
- Utilize recent work on optimizing Linux filesystems for flash storage (Arnd Bergmann’s work)
  - Ex: put meta-data on 4M block boundary, and have I/O request sizes match NAND block sizes

Resources

- How We Made Ubuntu Boot Faster – Scott Remnant, LinuxCon 2010
  - http://events.linuxfoundation.org/linuxcon2010/remnant
- Improving Android Boot-Up Time – Tim Bird, LinuxCon 2010
  - http://elinux.org/Improving_Android_Boot_Time
- Arnd Bergmann's work on optimizing cheap flash media
  - http://lwn.net/Articles/428584/
- Materials to support this presentation:
  - http://elinux.org/Android_Boot-time_Readahead

SONY
make.believe

Questions and Answers

Thanks!