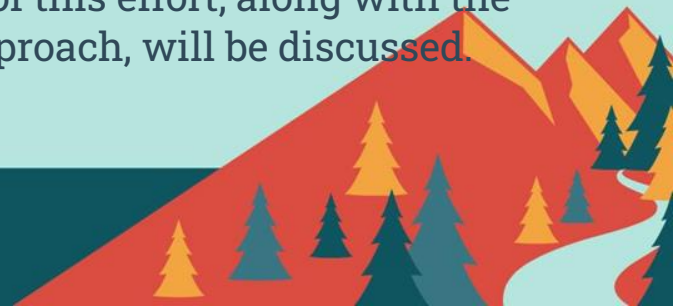
Regression Testing Boot-Time Performance in the Linux Kernel

Tim Bird
Principal Software Engineer, Sony

Abstract:

There are numerous tools to measure boot-time performance of Linux. However, there is no standard regression test of boot performance for Linux. This is due to a number of factors, including disparities in system performance, different requirements for quickly-needed functionality, and differences in bootloader, kernel and user-space configuration. In this session Tim will present a boot-time regression test that utilizes a collection of reference value data files for different platforms, kernel versions and configurations.

A meta-data matching system is used to select an appropriate reference data file. Boot time data (including initcall durations, and the durations of pre-selected boot operations) is compared with reference values, in order to report regressions in boot-time duration for specific elements of the boot sequence. The upstream status of this effort, along with the test and supporting tools, as well as issues found with this approach, will be discussed.



Outline

- Why no upstream boot-time regression test?
- My boot-time regression test
- Issues found
- Reference value files
- Meta-data matching
- Results
- Next steps

Why no upstream boot-time regression test?

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Why no boot-time test?

- I am only aware of 1 boot-time test for the Linux kernel
  - Laura Nao – ftrace-based, was not accepted upstream
- All other boot-time features and tools are instrumentation, inspection or visualization tools
  - They are NOT tests that yield a pass/fail result to be acted upon
  - In other words, boot-time testing is left as an exercise for the user (human)
- I am not aware of any automated performance tests of any kind in the upstream Linux kernel

Why no boot-time test? (continued)

- 2 parts to this absence:
  - Why is there no shared test?
  - Why is this non-thing not upstream?

Why no shared test?

- Boot-time code paths vary widely from machine to machine
  - Due to differences in kernel config, distribution, loaded drivers, boot environment
- Boot-time durations vary widely from machine to machine
  - Due to hardware differences (CPU speed and count, memory size, storage size, etc.)
- Items that need to come up quickly are different:
  - Usually a camera, in Automotive use cases
  - Could be something else for different embedded use cases
  - Even for same hardware type (camera), it may be connected to the system via different means: (e.g. a different bus: USB, MIPI, I2C, PCI, etc.), or different control bus (CAN, SPI, I2C, etc.)
  - This is similar to the "workload" in other benchmark tests – it's unique to your solution
- No single test can capture all use cases

Why no upstream test?

- A test requires consistent, generalizable pass/fail criteria
  - See previous slide
- Boot-time Specific issues:
  - Not all boot-time delays are within kernel scope
    - bootloader, firmware, other hardware, user space
  - Kernel boot-time blind spot
    - Portion of kernel boot occurs before clock and timer initialization
    - About 60 to 150 printk messages emitted with 0.000 timestamps
    - Representing about 100 to 400 milliseconds of boot time
    - This is a specific case of self-instrumentation not being there in time to provide useful data (same problem with early ftrace, or early console)
  - The best measurements would be from using external hardware
    - But that would be lab-specific

OK – Let's give up

Just kidding – Here's a Proposed solution

- An upstream test
- Reference values
  - Keep reference values separate from test
  - Store them somewhere outside of upstream
  - (optional) Automatically find a useful reference value file
- Automatic testing
  - With reporting to Maintainer, and/or contributor of change that caused regression

My Boot-Time Regression Test

Boot-time Regression Test

- Simple steps:
  - Measure values
  - Compare with Reference values
  - Report regressions (not done yet)
- Separate programs:
  - grab-boot-data.sh – for data collection
  - boot-time-regression-test.py – for comparing metrics with reference values
  - find-matching-ref-value.py – for automatically detecting a ref-value file

Measurement tool – grab-boot-data.sh

- Data collection:
  - Collects kernel boot messages (using dmesg)
  - Also collects meta-data for machine
    - uptime, kernel info, distro info, kernel config, free, df, mount, ps, /proc/cpuinfo
- Checks that that 'initcall_debug' was used on kernel command line
- Is a simple POSIX shell script
  - So it can run on ANY embedded Linux system that has coreutils (ps, dmesg, etc.)
- Saves boot-data file locally, but...
  - Also automatically sends the data to the 'boot-time' wiki
    - Uses 'curl'
    - But if curl or networking is missing, you can copy the file from an embedded device and upload it from a separate machine

grab-boot-data.sh usage

- You must specify a "lab" and "machine", so that your data can be distinguished from other data submitted to the boot-time wiki
- Example execution:
  - grab-boot-data.sh –l timslab –m rpi4
- Example result filename:
  - boot-data-timslab-bp1-250617-171816.txt
- Usage available with:
  - grab-boot-data.sh –h
- Tool itself and documentation are at:
  - htttps://birdcloud.org/boot-time/Boot-time_Tools

Measurement values

- What was measured?
  - initcalls
  - regions
    - Defined by printks, duration calculated by timestamp deltas
- Regions
  - Specified by a start and end printk for a region
    - Can have multiple starts or multiple ends
    - Can end on next printk
  - Have to establish each printk's region
    - e.g. memory init, network init, storage and filesystem init, individual drivers, etc.
  - Can accumulate values for activity types or subsystems
    - More than one printk span can contribute to a region duration
  - Was hard to find unique regex for some printks

Actual test: boot-time-regresson-test.py

- Processes a boot-data file produced by grab-boot-data.sh
- Compares results with a reference value file
  - The reference value file is another boot-data file
- Example usage (simple):
  - boot-time-regression-test.py boot-data-timslab-rpi4-250613-101432.txt \
    –r boot-data-timslab-rpi4-241109-153106-ref-values.txt
- Outputs results in KTAP format

Reporting regressions

- Deciding what constitutes a regression
  - Also known as the: "Pass/Fail criteria"
- Options:
  - result > ref value – too simple, lots of false positives
  - result > ref value +5% - still many false positives
  - result > ref value +5%, if delta>30 microseconds –weed out small changes
- Support for configurable pass/fail criteria?
  - That's what I recommended last year at Plumbers
    - I even wrote tools to handle criteria definitions in a separate file
  - For this boot-time regression test, I decided to keep it simple
    - There are less variables at boot time that affect performance from one boot to the next -> there should be less variation

boot-time results

```
KTAP version 1
1..760
# boot data file: boot-data-timslab-bp1-250617-171816.txt
# ref values file: boot-data-timslab-bp1-241106-170322-ref-values.txt
value con_init_duration=0 us
ok 1 initcall con_init
value hvc_console_init_duration=0 us
ok 2 initcall hvc_console_init
...
# reason: devlink_init_duration (4000) > ref value of '0' (delta=4000)
not ok 272 initcall devlink_init
...
value jent_mod_init_duration=32306 us
not ok 412 initcall jent_mod_init  # SKIP: reference value not found
...
# reason: phy_gmii_sel_driver_init_duration (687) > ref value of '546' (delta=114)
not ok 435 initcall phy_gmii_sel_driver_init
```

Test Issues

- Measurement issues:
  - Identifying boot regions (with printks) is hard
    - No common printks, sometimes no clear start or end to region
    - Might require adding additional printks (instrumentation points)
      - Not sure this is good for upstream (more printks = slower boot)
  - Aggregating durations hides where slowdown occurred
    - Is bad for reporting
  - Deferred probes disassociates delays
    - Can't correlate probe duration to correct region, initcall or driver
  - Timestamp blind spots gives lots of 0s – worthless data
  - Requires capturing data from a kernel booted with 'initcall_debug'
    - Test framework might not support modifying the kernel command line
      - I did a lot of manually reconfiguring boot scripts

Reference Values

Reference Value files

- Is a file containing the reference values to use for a test run
  - It has "known good" metrics for all test measurement values
  - You must have a reference value file to produce pass/fail test results
- Attributes:
  - Format
  - How Created
  - Where stored
- Finding the right reference values
  - Meta-data matching

Ref Values format

- Ref Values file is the same format as a boot data file
  - Consists of unmodified printk lines, with system meta-data
    - Could be a different, stripped-down format
    - Is just a boot data file from a different (/previous) run
  - Having same format as results file means it's easy to create
  - It's nice to keep the meta-data with the ref values file
  - Test already has a parser for the boot data file
    - No need for a separate parser
- How to create:
  - Copy or symlink a "good" boot-data file to include the string "-ref-values"
  - Example:
    - ln –s boot-data-timslab-bp1-241106-170322.txt \
      boot-data-timslab-bp1-241106-170322-ref-values.txt

Reference Value files location

- Local reference value files
  - Can be on local filesystem, but you have to specify the file
  - Allows you to run the test without accessing the internet
  - You have complete control over the reference values used
- Remote reference value files
  - Can be downloaded ahead of time, or at time of test
  - Can be automatically selected for you
  - Goal is to support testers who do not have experience or knowledge of 'known good' values

Reference Value file matching

- Test can use a local reference value file that you specificy
- Or you can use a tool to find one automatically
  - Tool can examine a list of pre-existing test results files
  - Tool can query the boot-time wiki for a good match
- Manually finding a ref-values match:
  - find-matching-ref-value-file.py
    - Is run on a set of boot data files
    - Reports scores for each one (optionally with reasons for the score)
    - You select the one to use, and specify it with the test
- boot-time-regression-test.py can automatically get a match from the boot-time wiki
  - Use the -r "remote-auto" option

Ref Values Issues

- Variance may be caused by random factors, leading to false positives
  - example: some durations are affected by timouts, which depend on when an initcall started relative to the timer tick
  - Could possibly mitigate by averaging results over multiple boots
  - Or, could allow higher variance for some items
- Requires another full parse of a boot data file
  - Simplified ref values file format would lead to faster test operation
  - Currently, speed of ref value parse is not an issue

Meta-data matching

Boot data files have meta-data

- boot data files (produced by grab-boot-data.sh) have lots of meta-data about the system of origin
- Meta-data is used for meta-data matching for ref value selection
- A single thing, like a changed kernel CONFIG, or a different kernel command line option, can affect the boot durations
  - Not adequate to just use data from same machine or type
  - Need other attributes the same also
- Need to make sure that test environment matches reference values environment

Libraries and tools for working with meta-data

- Python module: boot_data_parser.py
  - Parses and synthesizes meta-data from a boot-data file
  - Yeilds a python object with meta-data attributes
- Tool: show-boot-data.py – shows sets of meta-data or individual items
  - Can work on multiple files (good for comparing files)
- find-matching-ref-value-file.py
  - Generates a matching score based on weighted meta-data matches
  - machine, ARCH, config similarity, kernel cmdline, CPU count, bogomips, memory
- Server-side: ProcessorBootRegion.py
  - Allows for generating reports containing meta-data, including differences between boot data files

Example scoring output:

```
$ ./find-matching-ref-value-file.py boot-data-timslab-bp1-250123-184721.txt -d bd-files -s

boot-data-timslab-bp1-250123-184721.txt, score=72040
boot-data-timslab-bp1-241203-134136.txt, score=71240
boot-data-timslab-bp1-241106-170322.txt, score=71040
boot-data-timslab-bp1-250327-181829.txt, score=70990
boot-data-TI_bootLab-SK-AM62x-250221-062922.txt, score=66195
boot-data-francescoslab-beagleplay-250106-145753.txt, score=50773
boot-data-francescoslab-beagleplay-241230-221331.txt, score=50534
boot-data-francescoslab-imx93frdm-250429-065805.txt, score=50326
boot-data-timslab-rpi4-250612-052958.txt, score=31739
boot-data-manish_labs-beagleboneblack-700101-000057.txt, score=15964
boot-data-timslab-rpi3-241106-165048.txt, score=6674
boot-data-timslab-bbb-241106-233712.txt, score=-12246
boot-data-mhoyer-rpi_zero_w-250109-210016.txt, score=-13139
boot-data-timslab-timdesk-250611-224632.txt, score=-22864
boot-data-bobs_lab-sam1-241111-062218.txt, score=-23773
boot-data-timslab-xps13-250124-030801.txt, score=-25789
boot-data-timslab-xps13-250124-030311.txt, score=-26089
```

Scoring details

```
$ ./find-matching-ref-value-file.py boot-data-timslab-bp1-250123-184721.txt -d bd-files -v
...
boot-data-francescoslab-beagleplay-241119-111834.txt          51421 +500 for match on MACHINE of BeagleBoard.org BeaglePlay
                                                                     +5000 for match on ARCH of ARM64
                                                                     +57940 for same configs
                                                                     -1585 for config diffs
                                                                     -46 for (y vs m) config diffs
                                                                     -1075 for configs only in src bd
                                                                     -9213 for configs only in ref bd
                                                                     -50 for KERNEL_CMDLINE ref missing 'BOOT_IMAGE=/Image'
                                                                     -50 for KERNEL_CMDLINE ref missing 'log_buf_len=32M'
                                                                     -50 for KERNEL_CMDLINE ref adding 'LABEL=Boot'
                                                                     -50 for KERNEL_CMDLINE ref adding 'log_buf_len=10M'
                                                                     -50 for KERNEL_CMDLINE ref adding 'quiet'
                                                                     -250 for difference in 'quiet' arg
                                                                     -500 TOTAL for mis-matches in KERNEL_CMDLINE
                                                                     +200 for matching cpu counts (of 4)
                                                                     +200 for matching bogomips (of 1600.0)
...
```

Results

Results

- Findings
- Regressions found
- New tools and resources developed for this effort

Regressions for x86

- Desktop machine (Dell) going from 5.15.0 to 6.8.0
- ref-values were for:
  - the same hardware (timdesk)
  - running kernel version 5.15 (versus test on 6.8.0)
  - running Ubuntu 20.04 (versus test on 24.04)
- 10 initcalls were more than 10 ms longer
  - msr_init, jent_mod_init, pic_iommu_init, inet_init, coretemp_init, input_leds_init, joydev_init, intel_uncore_init, init_encrypted, vmd_drv_init

regression detection command

- $./boot-time-regression-test.py \
  bd-files/boot-data.*-timdesk-250611-224632.txt \
  -r bd-files/boot-data*timdesk*ref-values.txt \
  | grep delta | sort -n -t= -k2

initcalls with higher than 10 ms delta regression on x86 desktop machine

```
# reason: msr_init_duration (14138) > ref value of '1003' (delta=13135)
# reason: jent_mod_init_duration (26822) > ref value of '10246' (delta=16576)
# reason: pci_iommu_init_duration (20087) > ref value of '61' (delta=20026)
# reason: inet_init_duration (23963) > ref value of '3619' (delta=20344)
# reason: coretemp_init_duration (26748) > ref value of '1176' (delta=25572)
# reason: input_leds_init_duration (30115) > ref value of '541' (delta=29574)
# reason: joydev_init_duration (30134) > ref value of '14' (delta=30120)
# reason: intel_uncore_init_duration (30986) > ref value of '586' (delta=30400)
# reason: init_encrypted_duration (41183) > ref value of '7217' (delta=33966)
# reason: vmd_drv_init_duration (111725) > ref value of '19538' (delta=92187)
```

Regressions found

- Regions that have suffered regressions
  - Only have 1 initcall an ARM64 right now:
    - jent_mod_init (went from 30 ms to 50 ms)
    - Solution: don't use this module
    - Alternate Solution: Use deferred initcalls
      - (in progress): patch is out of tree
    - Possible other solutions:
      - Use config options to decrease entropy pool size or quality
      - Use boot cache to stuff with pre-calculated entropy
  - Expect to find more regressions with additional analysis
    - Need more data for better analysis of acceptable variance

Findings

- Defining boot regions from printks alone is hard
  - Not enough common printks
  - I originally intended to utilize only common printks
  - May switch to ftrace, but that adds overhead, and requires CONFIG_FTRACE AND kernel command line changes
- Need to drill down into regressing code (possibly using ftrace) to diagnose and fix regressions
  - Could possibly use git bisection or CONFIG bisection
  - Note that this is not part detecting the regression

Tools and Resources created

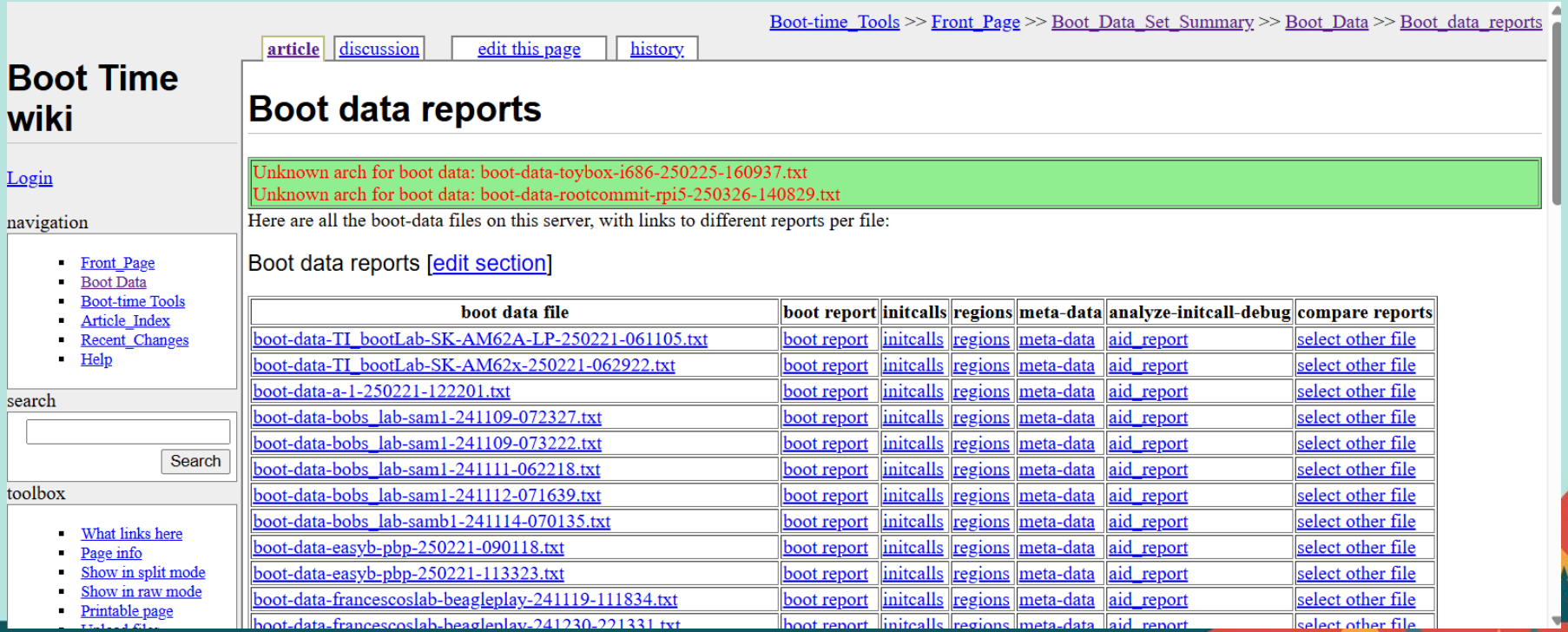
Tools and online resources built

- grab-boot-data.sh, boot-time-regression-test.sh, find-matching-ref-value-file.py, boot_data_parser.py
- Various tools to analyze meta-data and printk statements for region definitions
    - Including show-boot-data.py
- Show_delta – tool for calculating timestamp deltas
- Boot-time wiki (see next page)

The Boot-Time wiki

- Wiki with boot-data information
  - Storage of boot-data files and reference value files
- Also, visualization, analysis, and management tools
  - ProcessorBootRegion.py (cgi script for the boot-data wiki)
    - Allows for many different reports, including diff'ing two data sets
    - Shows summaries over multiple data sets:
      - Top 10 longest initcalls
      - Longest region durations
      - Duration of a single initcall across many systems, kernel versions, and test runs
    - Homebrew analytics is not great, but works
  - ProcessorRefFile
    - Handles queries for ref-values files (scoring, best match)
      - Using find-matching-ref-value-file.py on the back end
    - Supports management of designated ref values files on the wiki

Boot-time wiki boot data reports page



Boot-time_Tools >> Front_Page >> Boot_Data_Set_Summary >> Boot_Data >> Boot_data_reports

article | discussion | edit this page | history

Boot Time wiki

Login

navigation

- Front_Page
- Boot Data
- Boot-time Tools
- Article_Index
- Recent_Changes
- Help

search

Search

toolbox

- What links here
- Page info
- Show in split mode
- Show in raw mode
- Printable page

Boot data reports

Unknown arch for boot data: boot-data-toybox-i686-250225-160937.txt
Unknown arch for boot data: boot-data-rootcommit-rpi5-250326-140829.txt

Here are all the boot-data files on this server, with links to different reports per file:

Boot data reports [edit section]

boot data file	boot report	initcalls	regions	meta-data	analyze-initcall-debug	compare reports
boot-data-TI_bootLab-SK-AM62A-LP-250221-061105.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-TI_bootLab-SK-AM62x-250221-062922.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-a-1-250221-122201.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-bobs_lab-sam1-241109-072327.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-bobs_lab-sam1-241109-073222.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-bobs_lab-sam1-241111-062218.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-bobs_lab-sam1-241112-071639.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-bobs_lab-samb1-241114-070135.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-easyb-pbp-250221-090118.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-easyb-pbp-250221-113323.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-francescoslab-beagleplay-241119-111834.txt	boot report	initcalls	regions	meta-data	aid_report	select other file
boot-data-francescoslab-beagleplay-241230-221331.txt	boot report	initcalls	regions	meta-data	aid_report	select other file

Next Steps

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Next Steps

- Make github repo for test and related tools
- Gather boot-time data from more platforms, over more kernel versions
- Reporting: Need to automate reporting
  - Assign bug to sub-system and/or maintainer
  - Send report with details to mailing list (support opt-in for reports)
- Finish defining boot region definitions
- Better analysis of deferred probes
- Support extra-kernel regions, with Unified Boot Log
- Ultimate Goal: boot-time tuning tool
  - Automated tool to examine current boot-time, and recommend changes to improve
    - e.g. command-line args, config changes, probe ordering, deferred operations, etc.

Next Steps (cont.)

- Start reporting or fixing regressions
- Refine grab-boot-data.sh tool
  - Check for more error conditions
    - Missing CONFIG, too-small kernel log buffer, missing kernel symbols, etc.

Big Ask:

Please run grab-boot-data.sh
on your own system, to generate
more data for the boot-time wiki

Thanks

Contact me at: tim.bird@sony.com

To join the Boot-Time SIG activity, see https://elinux.org/Boot-Time_SIG

Extra Data
(unused slides)

Existing Instrumentation and Tools

Existing Instrumentation

- printk times
  - timestamp prefixed to every kernel message
- initcall_debug
  - Can add 'initcall_debug' to kernel command line, to cause extra printk output with:
    - initcall duration lines
    - probe duration lines
- ftrace
  - Can add tracing parameters to kernel command line, to trace specific functions or events

Existing Tools

- scripts/show_delta
- scripts/bootgraph.pl
- tools/power/pm-graph/bootgraph.py
- bootchart
- grabserial
- systemd-analyze
- ftrace
- analyze-initcall-debug.py

Problems with Performance Testing

Problems with Performance Testing

- For functional testing, answer is the same for every machine
  - pass, fail results
  - Sometimes report 'ignore', 'xfail' or something else for situations where a test doesn't apply
- For performance testing, each machine has a separate value that constitutes success or failure
  - Human has to evaluate results, or manually establish thresholds
  - Values change over time (as kernel improves, or as hardware changes)
  - Solution for regression testing is to refer to past results
- Often, performance is affected by numerous machine attributes
  - Results are "fuzzy", and results variations need to be supported
    - Usually done with delta thresholds (or variances)

Problems with Performance Testing (cont.)

- Different products have different requirements
  - Results depend on the hardware parameters, software configuration, and product requirements
  - No single value or value threshold works for everyone

Problem

- There are no automated performance tests in the upstream Linux kernel
  (that I am aware of)
- In particular, there is no boot-time regression test upstream
  - There isn't even just a boot-time test, let alone a regression test
- It doesn't help that there is no "boot-time" kernel maintainer