kernelci.org

Project overview and Collabora’s contribution

Guillaume Tucker

gtucker@collabora.com

28 Sep 2017
KernelCI: the project

- Build and boot Linux kernel revisions continuously
  - Detect and report failures almost in real time
- Community
  - Launched by Linaro, now has various contributors
- Mostly targeting embedded ARM-based platforms
  - Running mainline trees, not vendor kernels
- Simple tests that catch fundamental issues early
KernelCI: the infrastructure

- **KernelCI back-end**
  - web API
  - Mongo database

- **Jenkins**
  - web UI
  - web API

- **LAVA labs**
  - web API

- **git**
  - Linux source

- **KernelCI front-end**
  - web UI

1. **Linux source**
2. **Jenkins**
3. **LAVA labs**
4. **KernelCI back-end**
5. **KernelCI front-end**
6. **Smiley icon**
KernelCI: the test cycle

1. Some code lands in the Linux git repository
2. Jenkins triggers builds for various configurations
3. Jenkins triggers LAVA boots with the built kernels
4. Results are pushed to the KernelCI back-end
5. Results are available via the front-end and emails
6. Developers can fix issues and send new patches...
Labs

Cambridge

Kevin Hilman’s

Sjoerd’s
KernelCI: the labs

- Many various labs connected to KernelCI
  - Linaro, Collabora, BayLibre, Pengutronix...
  - Individuals: Kevin Hilman, Mark Brown...
- Linaro Automated Validation Architecture - LAVA
  - https://validation.linaro.org/static/docs/v2/index.html
- Not all labs use LAVA
  - target is to align everything with LAVA v2
  - some quirky devices are hard to automate
Collabora’s contribution

- LAVA lab
  - Up to 1000 KernelCI boot tests per day, ~20 boards
  - Sponsored VMs to run builds in Jenkins
- Enabled new device types (boards) in KernelCI
- Various contributions to LAVA
  - Bug fixes, IPv6, flashing images
- KernelCI core improvements
  - LAVA v2 callback support, automatic bisection
Automatic Bisection

• Some branches get merged with many commits
• When a boot fails, only a range of commits is known
• Goal: automatically isolate the breaking commit
• Challenges:
  - Avoiding false positives
  - Triggering relevant bisections (avoid duplicates...)
  - Deploying into production
  - Providing useful results
Automatic Bisection - current status

- Currently on staging, getting deployed:
  - Jenkins job to run a bisection using LAVA v2
  - Logic in the back-end to determine bisection ranges
  - Back-end triggers bisections after each boot report

- Things to do:
  - Store bisection results, not just boot results
  - Add checks to not run unwanted bisections
  - Test and bring to maturity
Automatic Bisection - what next?

- Make bisection results visible on the front-end
- Send email notifications with bisection results
- Include bisection of failing tests, not only boots
- Consolidate back-end API around regressions
- Add option to bisect against a different tree
  - For example, failure in linux-next against master
- Any other thoughts?
Other plans and controversial ideas

- Add more tests in addition to kselftest
  - See https://staging.kernelci.org/test/
  - hackbench, igt maybe for DRM/KMS coverage?
- New Android “modular kernel” trees?
- Add support for depthcharge (Chrome OS)
- Improve quality so people trust the boot results more
  - align boards’ bootloader configs
- Add support for multiple compilers
Thank you!