LISA & Friends
Linux Interactive System Analysis

Patrick Bellasi
patrick.bellasi@arm.com
Agenda
Presentation outline

- Short introduction of the main goals of the LISA toolkit
  What do we need and why?
- Overall view of the main framework components
- Example usage scenario
  Short introduction of a real (simple) use-case
  Interactive session with questions
Agenda
Presentation outline

- Short introduction of the main goals of the LISA toolkit
  What do we need and why?
- Overall view of the main framework components
- Example usage scenario
  Short introduction of a real (simple) use-case
  Interactive session with questions
Motivations
What is the aim of LISA[1]

A toolkit to support interactive analysis

- Supports the study of existing behaviours
  
  e.g. Helps with - “how the hell does this PELT thing work?”

- Supports the analysis of new code being developed
  
  e.g. What is the impact of code modifications on key behaviours ?

- Get insights on what's not working and possibly why

- Common language to share reproducible experiments
  
  Allows to reproduce experiments on different targets

  Flexible enough: programmers like extensible APIs

Motivations
Why (yet) another toolkit?

- Many different test suite already exist
  - KernelCI: mainly "just" build and boot validation... but a lot of it
  - LTP: “validate the reliability, robustness, and stability of Linux”
  - Intel's 0-day: continuous regression testing for mainline kernel

- These are mainly **black-box analyses** which do not give enough insights
  - Benchmarks **show regressions** but do **not pinpoint their reasons**
  - Brute force analysis can point just to a specific patch
    
    *Still just reports what code is broken but usually not why or how*
Motivations
What do we need?

- Simple yet powerful API to
  Generate test workloads and execute on test targets
  *Synthetic workloads allow to stimulate specific behaviours*
  Post process collected data to produce stats, plots and reports
  *A graphical representation is usually easy to understand than numbers*
  *A set of assertions on specific features are useful for further investigations*

- Main counter arguments
  *I can do everything with a bash scripts and some other tools*
  *LISA doesn't want to replace them, just make them (possibly) more easy to use*
Agenda
Presentation outline

- Short introduction of the main goals of the LISA toolkit
  What do we need and why?
- Overall view of the main framework components
- Example usage scenario
  Short introduction of a real (simple) use-case
  Interactive session with questions
Toolkit Organization
Abstract view of the flow

- Experimenting using an “interactive environment”
- Data analysis and post-processing
- Tests definitions to support regression analysis

Evaluate trade-offs on Power/Performances

Classical flow vs LISA flow
Toolkit Organization
Bird's eye view of the main components

Interactive test and analysis

Automated tests

Data Collection and Analysis

Hardware Abstraction Layer

https://github.com/ARM-software/trappy [3]
Agenda
Presentation outline

- Short introduction of the main goals of the LISA toolkit
  What do we need and why?
- Overall view of the main framework components
- Example usage scenario
  Short introduction of a real (simple) use-case
  Interactive session with questions
Example Usage Scenario
Analysis of a new Scheduler Feature

- Evaluate the SchedTune extension of the EA scheduler

  A task must run 30% of its time on a big CPU when boosted 15%

Interactive Session
Thank You!

Patrick Bellasi
patrick.bellasi@arm.com

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.
Detailed Examples
A bottom up presentation of all LISA modules
Main Components

IPython Notebooks: Interactive Python Scripting (and more)

▪ What is a Notebook?
  Web based interface for “interactive” code execution
    
  Code organized into cells which can be re-executed out-of-order
  
  Support for different languages and code completion
    
  Easy access to embedded documentation
  
  Key bindings available for all the main actions

▪ How can a notebook be used?
  Interactively build experiments
  
  Generate reports which can be exported in HTML
    
  Which mixes code and comments
  
  Export code as a standalone python script
Main Components
IPython Notebooks: Example

- Enter the LISA Shell
  Custom commands are available for most common operations

- Start the notebook server
  By default uses the local version of needed libraries
  
  *Easy access to the code of internal modules*

  *Thus you can easily contribute your patches back to the mainline ;-)***
Main Components

Devlib\(^1\): Target Abstraction

- Low-level library used by WorkloadAutomation
- Command execution is on the remote target
  - Supports multiple platforms: linux, android (and chromeos)
    - Using SSH or ADB as communication channels
    - Single connection for all commands
- Provides APIs for the main Linux frameworks
  - Generic modules: cgroups, cpufreq, cpuidle, hotplug, hwmon, thermal
  - Special modules: android, biglittle
- Support energy measurement instruments
  - TC2/Juno energy counters, ARM EnergyProbe, DAQs

\(^1\) https://github.com/ARM-software/devlib
\(^2\) https://github.com/ARM-software/workload-automation
Main Components
TestEnv: Test Environment setup for specific Targets

- In a nutshell: a wrapper of devlib
  - Simplifies code in notebooks and tests
  
  Provides the glue-code to setup a test environment
  
  E.g. connect to client, initialize modules, setup the output folder

  Allows the definition of the setup in a declarative format
  
  Could be either a file or an inline python dictionary

  Exposes the devlib API

- Provides additional APIs for some common tasks
  - E.g. deploy a different kernel, reboot the target
Main Components

WlGen: portable Synthetic Workloads generation

- Synthetic workloads configuration and execution
  
  `perf bench sched`

  *messaging (aka hackbench) and pipe*

  `rt-app`

  *set of base behaviours (periodic, step, ramp, ...) which can be composed to create more complex execution scenarios*

  *custom JSON configuration*

- Execution tunables support:
  
  CPU pinning, CGroups, FTrace
Main Components
Executor: tests configuration and data collection

- Simple automation for experimental data collection
- Using a simple dictionary or JSON configuration
  
  - `confs` target configurations to test
  - `wloads` synthetic workloads to execute on each configuration
  - `iterations` number of executions for each wload
Main Components
TRAPpy\textsuperscript{[1]}: From FTrace events to PANDAS DataFrames

- Based on PANDAS DataFrames
  
  *Python “standard” framework for data analysis and statistics*

  ftrace events are translated into tables

  *Events must match a specific template: (unique_word): ((key)=(value)) +

  Example (raw trace, i.e. generated by trace-cmd report -r):
  sudo-3224 [001] 228774.292951: sched_switch: prev_comm=sudo prev_pid=3224 prev_prio=120 prev_state=2048 next_comm=kschedfreq:1 next_pid=1822 next_prio=49

- API for trace event analysis

  Plots of *table:*key “signals”

  *both static and interactive plots*

  Provide data structure support for BART

[1] https://github.com/ARM-software/trappy
Data Analysis
Exploiting Platform Data for Trace Analysis

- Platform specific information can be useful
  e.g. CPU topology, OPP curves, EnergyModel data, …
  Information on these are collected by TestEnv
    `platform.json` file in the results folder (i.e. `te.res_dir`)

- TRAPpy is a generic module for trace events parsing
  It does not know about a specific platform
    `Even if this information are available via the LISA::TestEnv module`


although we can combine “on-demand” TRAPpy with platform data some commonly used analysis are worth to be shared
Data Analysis
Filtering and Plotting Predefined functions

- **LISA::Trace** glues platform data with TRAPpy DataFrames
  
  *more complete analysis dataset*

- **LISA::Filters**
  
  *commonly used events filtering functions*

- **LISA::TraceAnalysis**
  
  *commonly used trace events plots*

- **LISA::PerfAnalysis**
  
  *commonly used performance plots*
Data Analysis
Using RT-App to evaluate task performances

- RT-App extended to report performance metrics\(^1\)

\[ \text{MaxSlack} = \text{Period}_{\text{conf}} - \text{RunTime}_{\text{conf}} \]

\[ \text{PerfIndex} = \frac{\text{Period}_{\text{conf}} - \text{RunTime}_{\text{meas}}}{\text{MaxSlack}} \]

\[ \text{NegSlack}_{\text{percent}} = \frac{\sum \text{Max}(0, \text{RunTime}_{\text{meas}} - \text{Period}_{\text{conf}})}{\sum \text{RunTime}_{\text{meas}}} \]

suitable to evaluate some EAS behaviors

- optimal CPU/OPP selection and SchedTune boosting

- too pessimistic on single period missing
  - we will add an option to reset metrics after each new activation

- Other metrics can be added
  - Linaro proposed a "dropped-frames" counter,
    - we should integrate that as well

\[^1\] libs/utils/results.py::RTAppPerf
Automated Testing
LisaTest: Regression Testing Analysis

- Support for batch execution of tests
data collection driven by the lisa::executor module
  *easy to develop code on Notebook and than convert to a test*
config file based tests definition
  *a JSON file is used to describe “confs” and “wloads”*

- Tests executes after data collection complete
each test is defined within a function which name starts by "test_"

- Post processing and reporting functions available
Automated Testing
Evaluation of Energy-Performances trade-offs

- We can spend more energy provided that we get some performance benefits.
  SchedTune aims at controlling this trade-off at run-time.
- Experiments report Energy-vs-Performance metrics.

**Energy Delay Product (EDP)**

$$EDP = Energy \times \sum RunTime_{meas}$$
Automated Testing
BART\textsuperscript{[1]}: Behavioural Analysis

- Set of APIs on top of TRAPpy DataFrames
  allows to extract “features” from trace events
  
  \textit{How long a task run on a CPU? Does it switch to another CPU?}
  \textit{How long the temperature remain within a specified range?}

  Advanced tests for “sched switches” and “thermal events”
  \textit{the API is (going to be) generic enough to introduce other events}

- Aims at supporting the definition of behavioural tests
  small and self-contained functional behaviour
  \textit{e.g. task migration, frequency switch, OPP capping}

  a failure should pinpoint a specific code path
  \textit{suitable to evaluate the impact of code additions/updates}

\textsuperscript{[1]} https://github.com/ARM-software/bart