Harmonizing Open Source test definitions

Tim Bird
Fuego Test System Maintainer
Sr. Staff Software Engineer, Sony Electronics
Outline

• Differences between test frameworks
• What is the “test definition”
• Element-by-element comparison
  • Between Fuego and Linaro
• Ways to share definitions between Fuego and Linaro
## Test Framework differences

- Different test frameworks have different APIs, test models

<table>
<thead>
<tr>
<th></th>
<th>Fuego</th>
<th>LAVA</th>
<th>Yocto Project</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test driven from</strong></td>
<td>host</td>
<td>target</td>
<td>target</td>
</tr>
<tr>
<td><strong>target lifecycle</strong></td>
<td>multiple tests per boot</td>
<td>re-provision for every job</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>languages</strong></td>
<td>host: bash, python, yaml, json</td>
<td>sh, awk, yaml</td>
<td>python</td>
</tr>
<tr>
<td></td>
<td>target: sh</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>APIs</strong></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td><strong>dependencies</strong></td>
<td>permission, kconfig, mem, storage</td>
<td>permission, packages</td>
<td>packages</td>
</tr>
</tbody>
</table>
Different frameworks factor their data and services quite differently.

Where operations are performed:
- 1) central server, 2) on a local host, or 3) on-DUT

Which entity is responsible for performing the operation:
- 1) the test itself, 2) the framework, 3) an external service, or 4) the end user (tester)

When are operations performed:
- 1) during the test, 2) during post-processing, 3) synchronously, 4) asynchronously, etc.

Test definition elements are in different files, to support per-test, per-board, or per-lab customizations.
Fuego

Diagram key

- Boxes = processes or services
- Cylinders = repositories (persistent storage)
- Lines = APIs

- Lots of systems have implicit APIs or hardcoded values
  - e.g. save a raw file to local filesystem
What is a “Test definition”?

• Meta-data and instructions for a running a test
• Elements:
  • Information about a test
  • Pre-requisites and dependencies
  • Instructions for test execution
  • Output parsing
  • Test variables
  • Results analysis
  • Visualization control
Test definition survey

• Conducted survey in January
  • Results at: https://elinux.org/Test_definition_survey
• Tried to find some “Rosetta Stone” tests
  • That were the same in each system
  • Lots of fields are optional, so no single test revealed all elements
  • sysbench, OpenSSL, iperf
Files comparison

- **Fuego**
  - test.yaml – meta-data
  - fuego_test.sh – test functions, pre-requisites
  - spec.json – test variable definitions
  - parser.py – parser
  - criteria.json – pass criteria
  - chart_config.json – results formatting
  - `<tarfile>.tar.gz` – source for test program

- **Linaro**
  - sysbench.yaml – meta-data, test parameter definitions
  - sysbench.sh – test functions, dependencies, parser
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Meta-data

- Information about the test
- Mostly informational
  - Name
  - Description
  - License
- Some functional data
  - Test format version
  - Packaging manifest
Fuego:
Benchmark.sysbench/test.yaml

fuego_package_version: 1
name: Benchmark.sysbench
description: |
  Measure the performance of system operations
  This test only performs the 'cpu' performance test.
  There are other tests available.
license: GPL-2.0
author: Alexy Kopytov
maintainer: Tim Bird <tim.bird@sony.com>
version: 0.4.8
fuego_release: 1
type: Benchmark
tags: ['system', 'cpu', 'performance']
gitrepo: https://github.com/akopytov/sysbench
data_files:
  - fuego_test.sh
  - parser.py
  - spec.json
  - test.yaml
  - sysbench-0.4.8.tar.bz2
  - disable_libtool.patch
  - malloc_cross_compile_fix.patch

metadata:
  name: sysbench
  format: "Lava-Test-Shell Test Definition 1.0"
description: "SysBench is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive load. Current features allow to test fileio, cpu, memory, threads, mutex and oltp."
maintainer:
  - chase.qi@linaro.org
os:
  - [ 'debian', 'Ubuntu', 'fedora', 'centos', 'openembedded' ]
scope:
  - performance
environment:
  - lava-test-shell
devices:
  - [ 'hi6220-hikey', 'apq8016-sbc', 'mustang', 'moonshot', 'thunder', 'd03', 'd05' ]
params: [omitted]
run:
  steps:
    - cd ./automated/linux/sysbench/
    - /sysbench.sh -n "${NUM_THREADS}" -t "${TESTS}" -s "${SKIP_INSTALL}" 
    - ../../../utils/send-to-lava.sh ./output/result.txt

Linaro:
sysbench.yaml
# Meta-data Comparison Table

<table>
<thead>
<tr>
<th>test.yaml</th>
<th>ex. value</th>
<th>&lt;test&gt;.yaml</th>
<th>ex. value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuego_package_version</td>
<td>1</td>
<td>metadata:format</td>
<td>Lava-Test-Shell Test Definition 1.0</td>
<td>format indicator</td>
</tr>
<tr>
<td>name</td>
<td>Benchmark.sysbench</td>
<td>metadata:name</td>
<td>sysbench</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td></td>
<td>metadata:description</td>
<td></td>
<td>human description of test</td>
</tr>
<tr>
<td>license/author/version</td>
<td></td>
<td>-</td>
<td>-</td>
<td>test info details</td>
</tr>
<tr>
<td>maintainer</td>
<td></td>
<td>metadata:maintainer</td>
<td></td>
<td>person in charge of this test definition</td>
</tr>
<tr>
<td>fuego_version</td>
<td>1</td>
<td>metadata:version</td>
<td></td>
<td>release number for this definition</td>
</tr>
<tr>
<td>type/tags</td>
<td></td>
<td>scope</td>
<td>performance</td>
<td>for test classification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>os, devices</td>
<td></td>
</tr>
<tr>
<td>params</td>
<td>NTHREADS</td>
<td>params</td>
<td>NTHREADS</td>
<td>test variables</td>
</tr>
<tr>
<td>gitrepo</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>location of source</td>
</tr>
<tr>
<td>data_files</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>manifest for test package</td>
</tr>
<tr>
<td>(elsewhere)</td>
<td></td>
<td>run:steps</td>
<td></td>
<td>test instructions</td>
</tr>
</tbody>
</table>
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Pre-requisites

- Pre-conditions or state required for a test to execute
  - That cannot be changed
  - Something that prevents a test from executing at all
- Some examples:
  - Required kernel versions, kernel configuration
  - Required permissions (root, capabilities)
  - Required programs and libraries
  - Required capacity (e.g. minimum memory, processors, storage)
  - Required hardware (net device, bus, etc.)
    - Usually use kernel config as a proxy for this
  - Arbitrary system attribute (logging system, init system, distro)
- Confusing: Fuego calls this a dependency.
Pre-requisites

Fuego: fuego_test.sh
- Need variables:
  - NEED_MEMORY
  - NEED_FREE_STORAGE
  - NEED_KCONFIG
  - NEED_ROOT
  - NEED_PROGRAM
- Pre_check functions:
  - assert_define
  - is_on_target
  - is_on_sdk
  - assert_has_program

Linaro: <testname>.yaml, <testname>.sh
- os
- devices
- functions:
  - check_root
- packages
  - saw this in one script:

```bash
pkgs="git make docker"
for i in ${pkgs}; do
  if ! command -v "$i"; then
    error_msg "$i is required but not installed!"
  fi
done
```
- device tags?
Pre-requisites source

Fuego:

`Functional.openct/fuego_test.sh`

```bash
function test_pre_check {
  assert_has_program openct-control
  assert_has_program openct-tool
}
```

`Benchmark.hackbench/fuego_test.sh`

```bash
NEED_ROOT=1
function test_pre_check {
  assert_define BENCHMARK_HACKBENCH_PARAMS
}
```

Linaro:

`sysbench.sh`

```bash
! check_root && error_msg "Please run this script as root."
```
Dependencies

- Things to install in order to execute the program
  - packages, libraries, programs

- Fuego:
  - There’s no system support for this
  - But can put files from test_deploy(), and install during test_run()
    - No known instances of this

- Linaro:
  - install_deps – for installing required packages
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Test execution control

- Controls where and when and how long a test can execute
- Fuego:
  - Labels used to restrict job to a specific node (or node class)
  - Triggers left as an exercise for the user
    - Triggers defined by end user in Jenkins job
  - Timeouts come from testplans
- Linaro:
  - Test definition indicates acceptable target OS, device
  - Job definition indicates acceptable device tag?
  - Timeouts come from job definition?
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Test variables

• Test variables = parameters that can be modified to customize a test

• Fuego:
  • Described in test.yaml file
  • Defined in spec.json, or on ftc command line (dynamic variables)
  • Utilized in fuego_test.sh function test_run()

• Linaro:
  • Defined in <test>.yaml file, or in job or command line?
  • Utilized in run:steps, and in <test>.sh file
    • Seems common to parse them from <test>.sh command line
Test variables source

Fuego:
Benchmark.iperf3/test.yaml

params:
- server_ip:
  description: |
      IP address of the server machine.
  example: 192.168.1.45
  optional: yes
- client_params:
  description: extra parameters for the client
  example: -p 5223 -u -b 10G
  optional: yes

Benchmark.iperf3/spec.json

```json
{
  "testName": "Benchmark.iperf3",
  "specs": {
    "default": { "client_params": "-O 4 -t 64" },
    "zerocopy": { "client_params": "-O 4 -t 64 -Z" },
    "udp": { "client_params": "-t 60 -u -b 400M" }
  }
}
```

Linaro:
sysbench.yaml

params:
# "$\{TESTS\}" could be one or more of the following tests.
# TESTS: cpu memory threads mutex fileio oltp
# TESTS: "oltp"
TESTS: "percpu cpu memory threads mutex fileio"
# Number of threads to run.
# Run $\{nproc\} threads by default.
NUM_THREADS: "NPROC"
SKIP_INSTALL: "false"
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Test instructions

• Actual commands to execute on the device under test
• Fuego:
  • Instructions are in `fuego_test.sh:test_run()` function (but this runs on the host)
  • Uses ‘report’ to execute a command on a board and save output to log
• Linaro:
  • Instructions are in `<test>.yaml:run:steps`
  • And in `<test>.sh`, which runs on the device under test
Test Instructions

- **Fuego**: `fuego_test.sh` shell functions
  - `test_pre_check()`
  - `test_build()`
  - `test_deploy()`
  - `test_run()`
  - `test_processing()`
- Fuego divides test operations into pre-defined phases

- **Linaro**:
  - `<test>.yaml:run:steps`
  - `<test>.sh`
Test instructions source

Fuego:

Functional.hello_world/fuego_test.sh

tarball=hello-test-1.0.tgz
function test_build {
    make
}
function test_deploy {
    put hello $BOARD_TESTDIR/fuego.$TESTDIR/
}
function test_run {
    report "cd $BOARD_TESTDIR/fuego.$TESTDIR; \
        ./hello $FUNCTIONAL>Hello_WORLD_ARG"
}
function test_processing {
    log_compare "$TESTDIR" "1" "SUCCESS" "p"
}

Linaro:

sysbench.yaml

run:
    steps:
        - cd ./automated/linux/sysbench/
        - ./sysbench.sh -n "${NUM_THREADS}" -t "${TESTS}" \
            -s "${SKIP_INSTALL}" \
            - ../../../utils/send-to-lava.sh ./output/result.txt

sysbench.sh

#!/bin/sh -e
   .../..../lib/sh-test-lib

OUTPUT="$\(pwd\)/output"
RESULT_FILE="${OUTPUT}/result.txt"
...
install_sysbench() ...
general_parser() ...
for tc in ${TESTS}; do
    ...
        sysbench --num-threads="${NUM_THREADS}" \ 
            --test="${tc}" run | tee "${logfile}".
    ...

...
Build instructions source

Fuego:
Benchmark.sysbench/fuego_test.sh

install_sysbench() {
    git clone https://github.com/akopytov/sysbench
    cd sysbench
    git checkout 0.4
    ./autogen.sh
    if echo "${TESTS}" | grep "oltp"; then
        ./configure
    else
        ./configure --without-mysql
    fi
    make install
    cd ../
}

Linaro:
sysbench.sh

function test_build {
    patch -p0 < $TEST_HOME/malloc_cross_compile_fix.patch
    patch -p0 < $TEST_HOME/disable_libtool.patch
    ./autogen.sh

    cp /usr/share/misc/config.{sub,guess} config
    ./configure --host=$PREFIX --without-mysql
    make
}

tarball=sysbench-0.4.8.tar.bz2
## Test instructions (and other stuff)

### Comparison Table

<table>
<thead>
<tr>
<th><strong>Fuego</strong></th>
<th><strong>Notes</strong></th>
<th><strong>Linaro</strong></th>
<th><strong>Notes</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>test_pre_check</td>
<td></td>
<td>check_root</td>
<td></td>
</tr>
<tr>
<td>test_build</td>
<td>tarball or git repo, cross-build on host</td>
<td>install_sysbench</td>
<td>git clone, build on target</td>
</tr>
<tr>
<td></td>
<td></td>
<td>install_sysbench</td>
<td>install required packages</td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test_deploy</td>
<td>put</td>
<td>install_sysbench</td>
<td>make install</td>
</tr>
<tr>
<td>test_run</td>
<td></td>
<td>run:steps/sysbench.sh</td>
<td></td>
</tr>
<tr>
<td>parser.py</td>
<td>python parsing</td>
<td>general_parser</td>
<td>sh, sed, awk</td>
</tr>
<tr>
<td>test.yaml:params</td>
<td>describe parameter vars</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>spec.json</td>
<td>define parameter values</td>
<td>&lt;test&gt;.yaml:params</td>
<td>define parameter values</td>
</tr>
<tr>
<td>test.yaml:data_files</td>
<td>manifest for packaging</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>criteria.json</td>
<td>results analysis</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis
- Visualization control
Parsing

- Parsing is:
  - Extracting testcase and measurement results from test output
- Fuego:
  - Performed by parser.py, using a library of helper functions
  - Also, data points are converted to a common format (run.json)
  - Also, charting data (html tables and plots) are prepared
- Linaro:
  - Performed by <test>.sh (to get to data points)
    - data points converted to LAVA-parsable strings in console output
  - ?? – converts data points into common format
  - ?? – converts common format into charts
```python
import os, re, sys
measurements = {}
regex_string = "^.*total time:\s+([d]{1,8}.?[d]{1,43})s$"
matches = plib.parse_log(regex_string)
if matches:
    measurements['cpu.prime_calculation'] = [{'name": "time", "measure": float(matches[0])}]
regex_string = "^.*events \((avg/stddev)\):s+([d]{1,8}.?[d]{1,43})/.*$"
matches = plib.parse_log(regex_string)
if matches:
    measurements['cpu.thread_fairness'] = [{'name": "events", "measure": float(matches[0])}]
sys.exit(plib.process(measurements))
```

```bash
Fuego: Parsing source
Benchmark.sysbench/parser.py
```

```bash
Linaro: sysbench.sh
general_parser() {
    # if $1 is there, let's append to test name in the result file
    local tc="$tc$1"
    ms=$(grep -m 1 "total time" "$logfile" | awk '{print substr($NF,1,length($NF)-1)}')
    add_metric "$tc-total-time" "pass" "$ms" "s"
    ms=$(grep "total number of events" "$logfile" | awk '{print $NF}')
    add_metric "$tc-total-number-of-events" "pass" "$ms" "times"
    ...
    ms=$(grep "execution time (avg/stddev)" "$logfile" | awk '{print $NF}')
    ms_avg=$(echo "$ms" | awk -F'|' '{print $1}')
    ms_stddev=$(echo "$ms" | awk -F'|' '{print $2}')
    add_metric "$tc-execution-time-avg" "pass" "$ms_avg" "s"
    add_metric "$tc-execution-time-stddev" "pass" "$ms_stddev" "s"
}
```
• Location and factoring are different:
• Location:
  • Fuego does all parsing on the host
  • Linaro does testcase and measurement conversion to a common format, on the target
  • Linaro does conversion to common format on ??? (where?)
• Factoring:
  • Fuego does extraction, conversion to common format, chart preparation in one step
  • Linaro does some of these operations elsewhere
Linaro deprecated their “parse pattern and fixup dictionaries”, with the following quote:

“Parse patterns and fixup dictionaries are confusing and hard to debug.”

source: https://lkft.validation.linaro.org/static/docs/v2/lava_test_shell.html
Elements Comparison

- Information about a test (meta-data)
- Pre-requisites and dependencies
- Execution control
- Test variables
- Instructions for test execution
- Output parsing or conversion
- Results analysis (skipping these)
- Visualization control
Linaro things with no Fuego analog

- Package dependencies
- os
- devices
- Multi-node tests
  - Every Fuego test is implicitly multi-node with the host as the other node
Fuego things with no Linaro analog

- Results analysis - Pass criteria
- Visualization control - Chart config
- Test package
- Target package
Results Analysis

- Describes the requirements (pass counts, fail counts, fail-ok-lists, benchmark value thresholds) for test to pass
  - Used for automated test interpretation
  - This determines the ultimate ‘red or green’ result
  - List of expected failures, or results that are ignored for now
- In criteria.json file
- Able to customize per board, or by some other attribute
- Example: LTP
  - raspberry pi has 28 failures
  - beagleone black has 67 failures, 2 hangs, and 1 kernel panic
    - (kernel panic and hangs testcases are potential skiplist items)
Visualization control

- Control over results output selection and formatting
  - Indicate table or plot
  - Select data points for output
- In chart_config.json file
- (Planned) report templates
Test package

- Package of test materials for sharing with others
  - Consists of metadata, source, instructions, parser, etc.
- Can be sent to a central server, for dispatch to another lab
- Central server feature is not finished yet
Target package

• Package of binary materials to be installed on a board
  • pre-built binaries, board-side scripts
  • Result of ‘deploy’ operation
• May correspond to an ‘image’
  • Is not the Software Under Test
• *Central server feature is not finished yet*
Ideas for harmony

- Migrate our test definitions to use same elements
- Run Fuego tests in LAVA
- Run Linaro tests in Fuego
- Co-develop APIs for lab hardware
**Migrate test definitions**

- Migrate our test definitions to use same elements
- Use same pre-requisite and dependency names
- Use common test variable names and values
- Regularize the test phases in Linaro scripts
- Create a common library for shared features
  - Fuego core function library, and fuego_board_function_lib.sh
  - Linaro sh-test-lib
Run Fuego tests in LAVA

- Old method:
  - Get ssh connection to board from LAVA
  - Have Fuego do it’s thing over ssh
  - Doesn’t integrate with LAVA scheduler or results processing

- New plan:
  - Write a converter to run a Fuego test in LAVA
    - Deploy target package to LAVA image server?
    - Requires to execute instructions that normally run on Fuego host on the Device under test
      - Maybe create or port the Fuego core library
      - Maybe just run Fuego CLI??

- Issues:
  - Fuego instructions for host are allowed to use bash-isms
Run Linaro tests in Fuego

- Already have a prototype:
  - Functional.linaro
  - Uses test-runner to send a job to LAVA
- Issues:
  - ???
Co-develop APIs for lab hardware

- Start with pdudaemon for PDU control
- Develop consistent APIs for other board control functions:
  - pressing buttons/flipping jumpers
  - bus control (ex dropping vbus to USB)
  - audio collection
  - video collection
  - power monitoring
Harmonization issues

- Factoring of test definition elements
  - Time/location of execution
  - What files has each element
  - Concepts that one system has that the other doesn’t
- Required software on target
- Required hardware in lab
- Results format (but that’s a whole other topic)
  - But: test definition defines testcase and metric names!
  - Wildly different in Fuego and Linaro, even for the same test
Let’s discuss the details....
Thanks

Tim Bird
Fuego Test System Maintainer
Sr. Staff Software Engineer, Sony Electronics