

Regression Testing and Continuous Integration with ReFrame

CSCS User Lab Day – Meet the Swiss National Supercomputing Centre

Vasileios Karakasis, CSCS

September 11, 2018

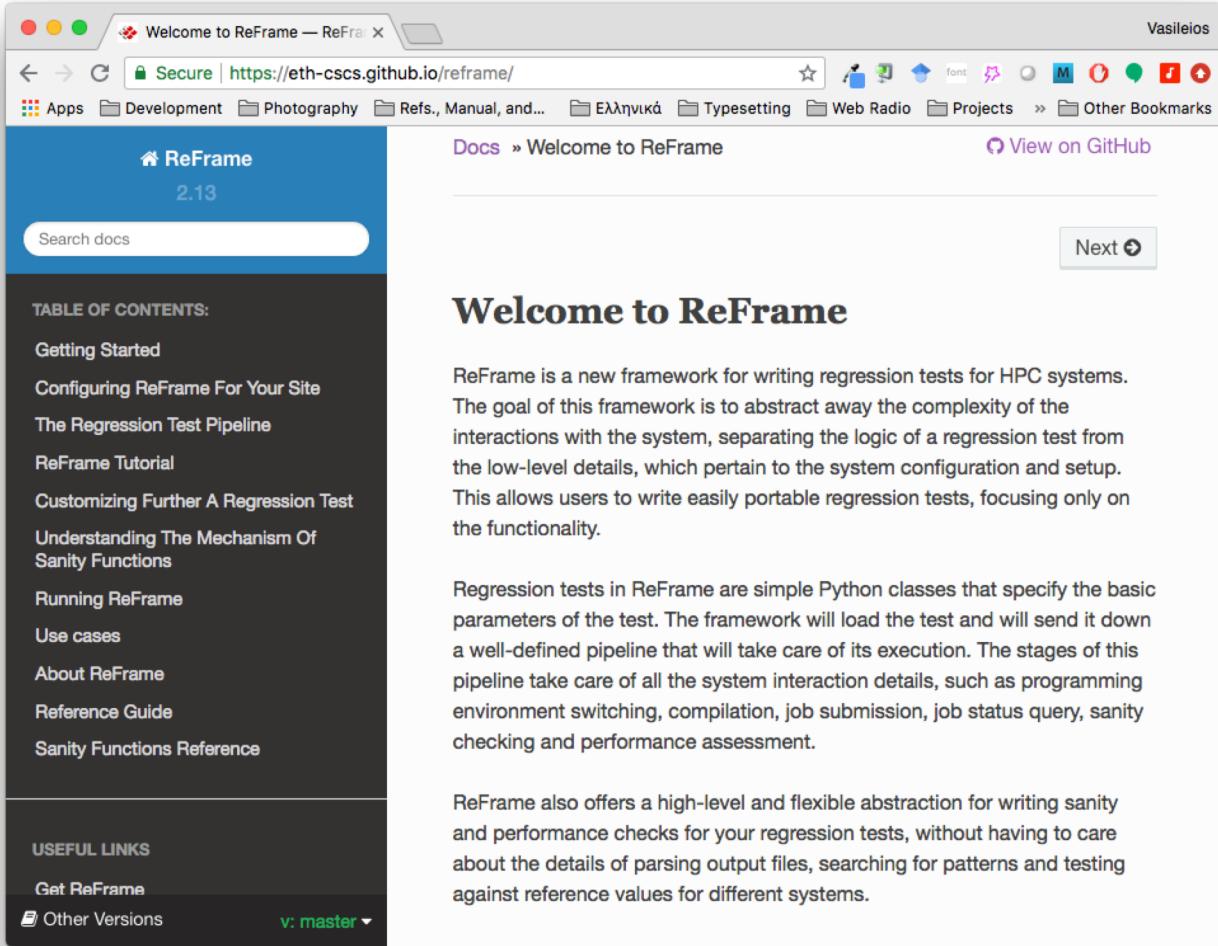
Table of Contents

1. Overview of ReFrame
2. Using ReFrame with CSCS' CI infrastructure

What is ReFrame?

A new regression testing framework that

- allows writing **portable HPC** regression tests in Python,
- **abstracts away** the system interaction details,
- lets users focus solely on the **logic** of their test.



The screenshot shows a web browser displaying the official ReFrame documentation at <https://eth-cscs.github.io/reframe/>. The page title is "Welcome to ReFrame — ReFrame 2.13". The left sidebar contains a "TABLE OF CONTENTS" with links to various documentation sections: Getting Started, Configuring ReFrame For Your Site, The Regression Test Pipeline, ReFrame Tutorial, Customizing Further A Regression Test, Understanding The Mechanism Of Sanity Functions, Running ReFrame, Use cases, About ReFrame, Reference Guide, and Sanity Functions Reference. Below the table of contents is a "USEFUL LINKS" section with links to "Get ReFrame" and "Other Versions". The main content area is titled "Welcome to ReFrame" and describes the framework's purpose: "ReFrame is a new framework for writing regression tests for HPC systems. The goal of this framework is to abstract away the complexity of the interactions with the system, separating the logic of a regression test from the low-level details, which pertain to the system configuration and setup. This allows users to write easily portable regression tests, focusing only on the functionality." It also explains that "Regression tests in ReFrame are simple Python classes that specify the basic parameters of the test. The framework will load the test and will send it down a well-defined pipeline that will take care of its execution. The stages of this pipeline take care of all the system interaction details, such as programming environment switching, compilation, job submission, job status query, sanity checking and performance assessment." At the bottom right of the content area, there is a link to "View on GitHub".

<https://github.com/eth-cscs/reframe>

Design Goals

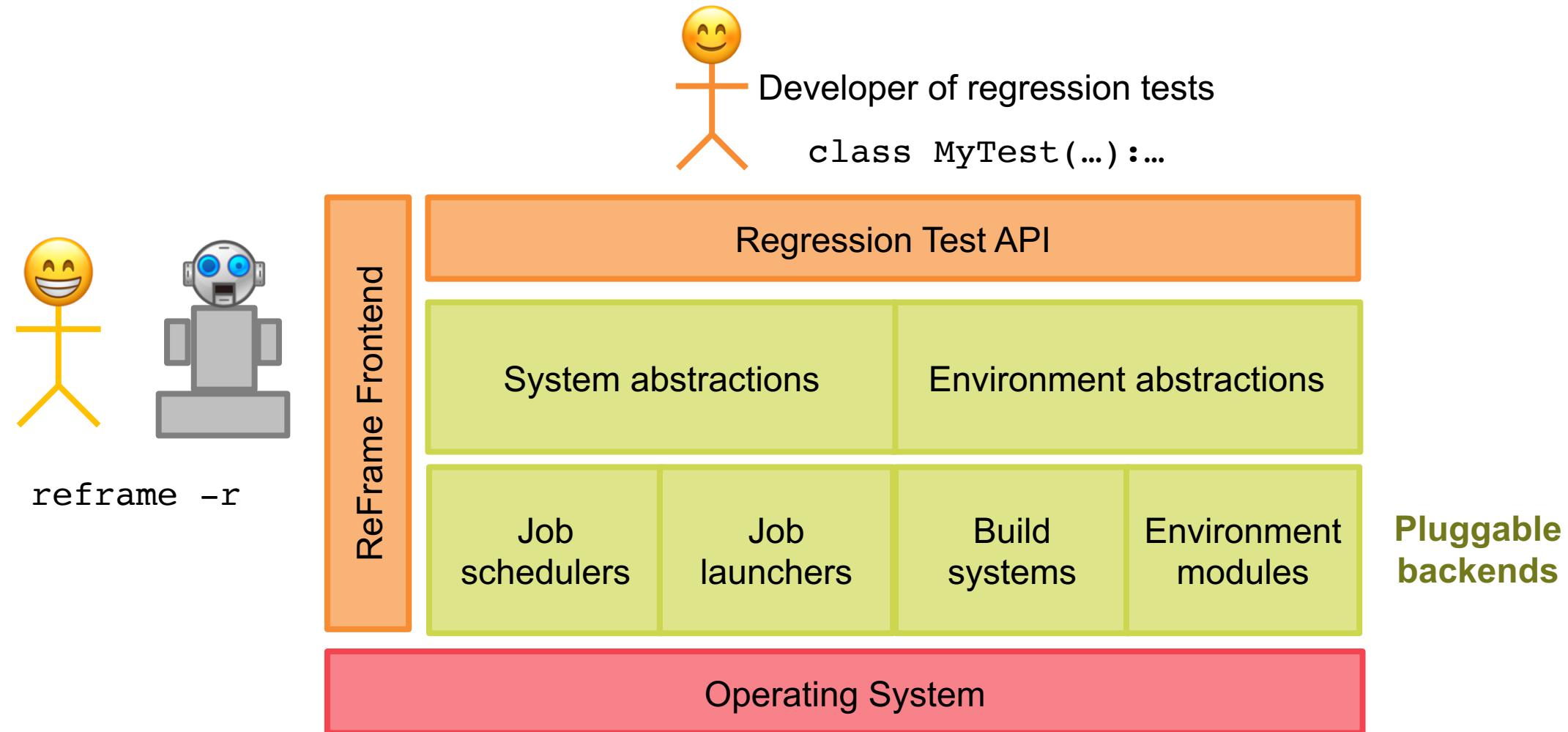
- Productivity
- Portability
- Speed and Ease of Use
- Robustness

Write once, test everywhere!

Key Features

- Separation of system and prog. environment configuration from test's logic
- Support for cycling through prog. environments and system partitions
- Regression tests written in Python
 - Easy customization of tests
 - Flexibility in organizing the tests
- Support for sanity and performance tests
 - Allows complex and custom analysis of the output through an embedded mini-language for sanity and performance checking.
- Progress and result reports
- Performance logging with support for Graylog
- Clean internal APIs that allow the easy extension of the framework's functionality
- ... and more (<https://github.com/eth-cscs/reframe>)

ReFrame's architecture



Writing a Regression Rest in ReFrame

A regression test writer should not care about...

- How to access to system partitions and if there are any.
- How (programming) environments are switched.
- How the test's environment is actually set up.
- How a job script is generated and if it's needed at all.
- How a sanity/performance pattern is looked up in the output.

ReFrame allows you to focus on the logic of your test.

Writing a Regression Test in ReFrame

```
.../tutorial/example7.py [reframe]
reframe> tutorial> example7.py >
example7.py x
1 import reframe as rfm
2 import reframe.utility.sanity as sn
3
4
5 @rfm.simple_test
6 class Example7Test(rfm.RegressionTest):
7     def __init__(self):
8         super().__init__()
9         self.descr = 'Matrix-vector multiplication (CUDA performance test)'
10        self.valid_systems = ['daint:gpu']
11        self.valid_prog_environ = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
12        self.sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
13        self.build_system = 'SingleSource'
14        self.build_system.cxxflags = ['-O3']
15        self.executable_opts = ['4096', '1000']
16        self.modules = ['cudatoolkit']
17        self.num_gpus_per_node = 1
18        self.sanity_patterns = sn.assert_found(r'time for single matrix vector multiplication', self.stdout)
19        self.perf_patterns = {
20            'perf': sn.extractsingle(r'Performance:\s+(?P<Gflops>\S+)\s+Gflop/s', self.stdout, 'Gflops', float)
21        }
22        self.reference = {
23            'daint:gpu': {'perf': (50.0, -0.1, 0.1)}
24        }
25        self.maintainers = ['you-can-type-your-email-here']
26        self.tags = {'tutorial'}
```

ReFrame tests are specially decorated classes

Valid systems and prog. environments for this test

Compilation and run setup

Sanity checking

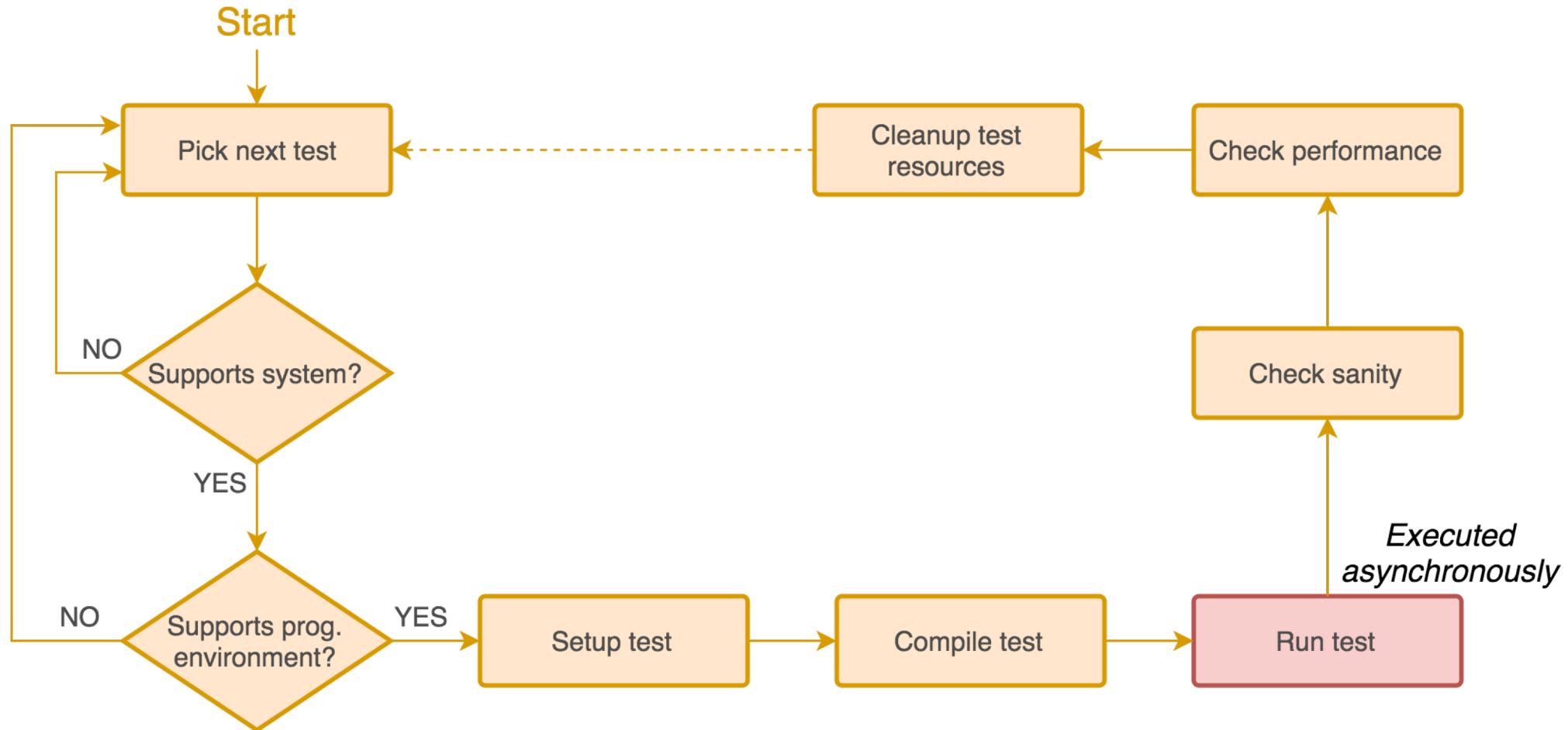
Extract performance values from output

Reference values and perf. thresholds

Tag test for easy lookup

The Regression Test Pipeline / How ReFrame Executes Tests

A series of well defined phases that each regression test goes through



The Regression Test Pipeline / How ReFrame Executes Tests

- Tests may skip some pipeline stages
 - Compile-only tests
 - Run-only tests
- Users may define additional actions before or after every pipeline stage by overriding the corresponding methods of the regression test API.
 - E.g., override the setup stage for customizing the behavior of the test per programming environment and/or system partition.
- Frontend passes through three phases and drives the execution of the tests
 1. Regression test discovery and loading
 2. Regression test selection (by name, tag, prog. environment support etc.)
 3. Regression test listing or execution

Running ReFrame

```
reframe -C /path/to/config.py -c /path/to/checks -r
```

- ReFrame uses three directories when running:
 1. **Stage directory**: Stores temporarily all the resources (static and generated) of the tests
 - Source code, input files, generated build script, generated job script, output etc.
 - This directory is removed if the test finishes successfully.
 2. **Output directory**: Keeps important files from the run for later reference
 - Job and build scripts, outputs and any user-specified files.
 3. **Performance log directory**: Keeps performance logs for the performance tests
- ReFrame generates a summary report at the end with detailed failure information.

Running ReFrame (sample output)

```
[=====] Running 1 check(s)
[=====] Started on Fri Sep  7 15:32:50 2018

[-----] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN   ] Example7Test on daint:gpu using PrgEnv-cray
[    OK  ] Example7Test on daint:gpu using PrgEnv-cray
[ RUN   ] Example7Test on daint:gpu using PrgEnv-gnu
[    OK  ] Example7Test on daint:gpu using PrgEnv-gnu
[ RUN   ] Example7Test on daint:gpu using PrgEnv-pgi
[    OK  ] Example7Test on daint:gpu using PrgEnv-pgi
[-----] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[  PASSED ] Ran 3 test case(s) from 1 check(s) (0 failure(s))
[=====] Finished on Fri Sep  7 15:33:42 2018
```

Running ReFrame (sample failure)

```
[=====] Running 1 check(s)
[=====] Started on Fri Sep  7 16:40:12 2018

[-----] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN   ] Example7Test on daint:gpu using PrgEnv-gnu
[ FAIL  ] Example7Test on daint:gpu using PrgEnv-gnu
[-----] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[ FAILED ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[=====] Finished on Fri Sep  7 16:40:22 2018
```

SUMMARY OF FAILURES

FAILURE INFO for Example7Test

- * System partition: daint:gpu
 - * Environment: PrgEnv-gnu
 - * Stage directory: /path/to/stage/daint/gpu/PrgEnv-gnu/Example7Test
 - * Job type: batch job (id=823427)
 - * Maintainers: ['you-can-type-your-email-here']
 - * Failing phase: performance
 - * Reason: sanity error: 50.363125 is beyond reference value 70.0 (l=63.0, u=77.0)
-

Running ReFrame (examining a failure)

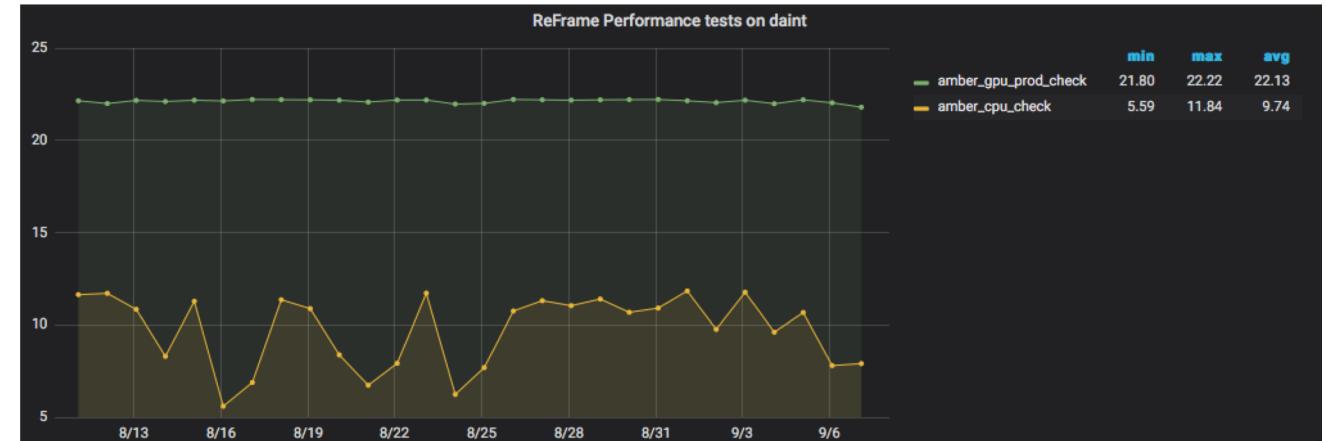
- ReFrame executes each test case from a separate stage directory:
 - `/path/to/stage/<system>/<partition>/<testname>/<environ>`
- Auto-generated build script and compilation's standard output/error
 - `rfm_<testname>_build.sh`
 - `rfm_<testname>_build.out`
 - `rfm_<testname>_build.err`
- Auto-generated job script and execution's standard output/error
 - `rfm_<testname>_job.sh`
 - `rfm_<testname>_job.out`
 - `rfm_<testname>_job.err`

Running ReFrame (examining performance logs)

- `/path/to/reframe/prefix/perflogs/<testname>.log`
 - A single file named after the test's name is updated every time the test is run
 - Log record output is fully configurable

```
2018-09-07T15:32:59|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-cray|jobid=823394|perf=49.71432|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T15:33:11|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-gnu|jobid=823395|perf=50.1609|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T15:33:42|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-pgi|jobid=823396|perf=51.078648|ref=50.0 (l=-0.1, u=0.1)
2018-09-07T16:40:22|reframe 2.14-dev2|Example7Test on daint:gpu using PrgEnv-gnu|jobid=823427|perf=50.363125|ref=70.0 (l=-0.1, u=0.1)
```

- ReFrame can also send logs to a Graylog server, where you can plot them with web tools.



Using ReFrame with the CSCS CI service

Use Case

- Target code: Arbor Library
 - A library for implementing performance portable network simulations of multi-compartment neuron models.
 - <https://github.com/eth-cscs/arbor>
- Goals
 - Run the library's unit tests with...
 - different supported compilers (GNU, Clang)
 - different code configurations (base, GPU-enabled, SIMD-enabled)
 - Reuse the same ReFrame tests for other target systems
 - My laptop (where Clang is available)

Limitations

- The Jenkins VM is allowed to run only sbatch
- VM's environment is completely different from the target environment (Daint)

ReFrame cannot be run directly from the Jenkins VM, but...

we can configure it properly for Daint's compute nodes and sbatch it.

Configuration Steps

1. Set up the Jenkins project (as shown previously)
2. Prepare the code repository
 - Add a Jenkinsfile at the top-level directory
 - Add a batch script for running ReFrame on the compute nodes
 - Add the ReFrame tests
 - Add a ReFrame configuration file for the target systems
3. Open a pull request

Code Repository Set-up

Branch: ci/reframe-test... ▾ arbor / ci /

Create new file Upload files Find file History

This branch is 15 commits ahead, 3 commits behind eth-cscs:master.

vkarak Fix Jenkinsfile Latest commit 8f7889e 2 days ago

..

arbor_tests.py Add Clang tests 2 days ago

cscs-daint-gpu.sh Fix Jenkinsfile Add Clang tests 2 days ago

rfm-config.py Add Clang tests 2 days ago

ReFrame tests

Batch script to launch ReFrame on the compute nodes

ReFrame config

File	Commit Message	Time Ago
arbor_tests.py	Add Clang tests	2 days ago
cscs-daint-gpu.sh	Fix Jenkinsfile Add Clang tests	2 days ago
rfm-config.py	Add Clang tests	2 days ago

Demo branch: <https://github.com/vkarak/arbor/tree/ci/reframe-tests/ci>

ReFrame Configuration

- Daint
 - Job scheduler: `local+srun` (i.e., do not submit any job, but use `srun` to parallel launch executables)
 - Programming environments: Cray, GNU, Intel, PGI
- My laptop
 - Job scheduler: `local` (i.e., do not submit any job and do not use any parallel launcher)
 - Programming environments: Clang
- Other systems (such as TravisCI's VMs) etc.

ReFrame Tests

```
68 @rfm.parameterized_test(['haswell'], ['broadwell'], ['native'])
69 class ArborSIMDTest(ArborBaseTest):
70     def __init__(self, arch_kind):
71         super().__init__()
72         if arch_kind == 'haswell':
73             self.valid_systems = ['daint:gpu']
74         elif arch_kind == 'broadwell':
75             self.valid_systems = ['daint:mc', 'tresas']
76         elif arch_kind == 'native':
77             self.valid_systems = ['tresas']
78
79         self.arch_kind = arch_kind
80
81     @property
82     def cmake_options(self):
83         return ['-DARB_WITH_ASSERTIONS=ON',
84                 '-DARB_VECTORIZE=ON',
85                 '-DARB_ARCH=%s' % self.arch_kind]
```

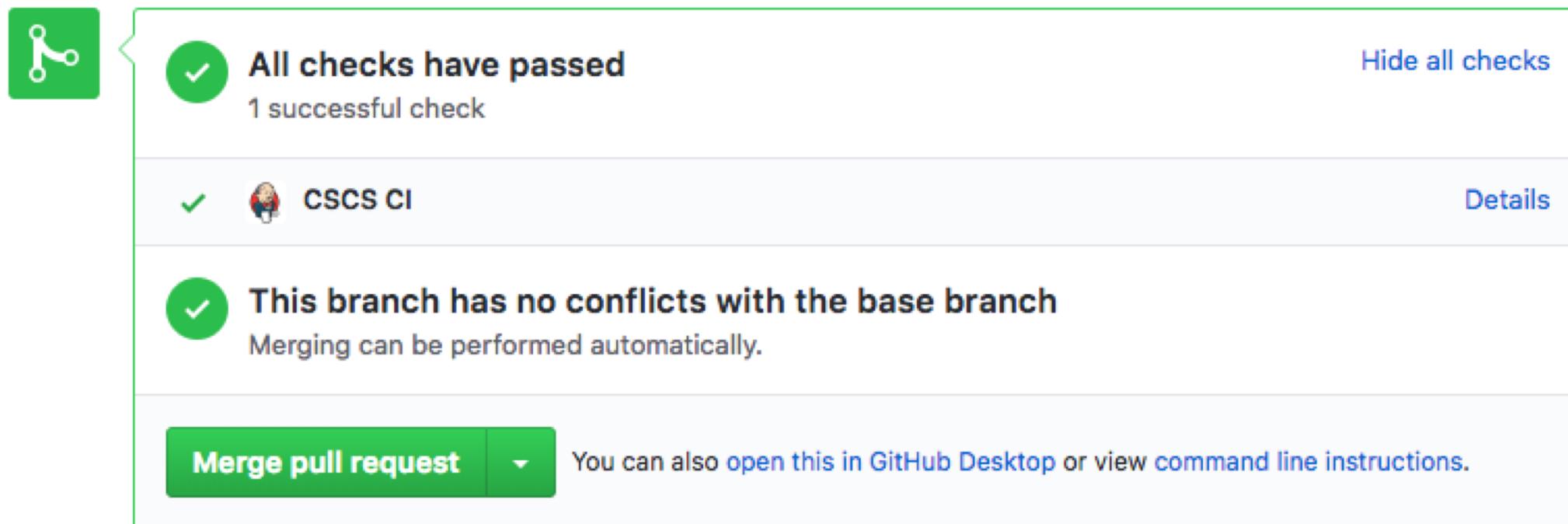
Use parameterized tests
to create test factories!

You can use inheritance to avoid
redefining common functionality!

```
66     rBaseTest):
67
68     def __init__(self):
69         super().__init__()
70
71         self.valid_systems = ['daint:gpu', 'daint:mc']
72         self.valid_environments = ['PrgEnv-gnu']
73
74
75     @property
76     def cmake_options(self):
77         return ['-DARB_WITH_ASSERTIONS=ON',
78                 '-DARB_VECTORIZE=ON',
79                 '-DARB_ARCH=%s' % self.arch_kind]
80
81     @property
82     def job_options(self):
83         return {'C_COMPILER': environ.cc,
84                 'CXX_COMPILER': environ.cxx,
85                 'CMAKE_C_FLAGS': config_opts['CFLAGS'],
86                 'CMAKE_CXX_FLAGS': config_opts['CXXFLAGS']}
87
88     def run(self, partition, environ, **job_opts):
89         cmd = [self._cmd, '-j', str(self._nproc),
90                '-v', '-DARB_WITH_ASSERTIONS=ON']
91
92         if self._nproc > 1:
93             cmd.append('--parallel')
94
95         cmd.extend(job_opts['CMAKE_C_FLAGS'])
96
97         if self._nproc > 1:
98             cmd.append('--parallel')
99
100            .join(config_opts)
101
102            p(partition, environ, **job_opts)
```

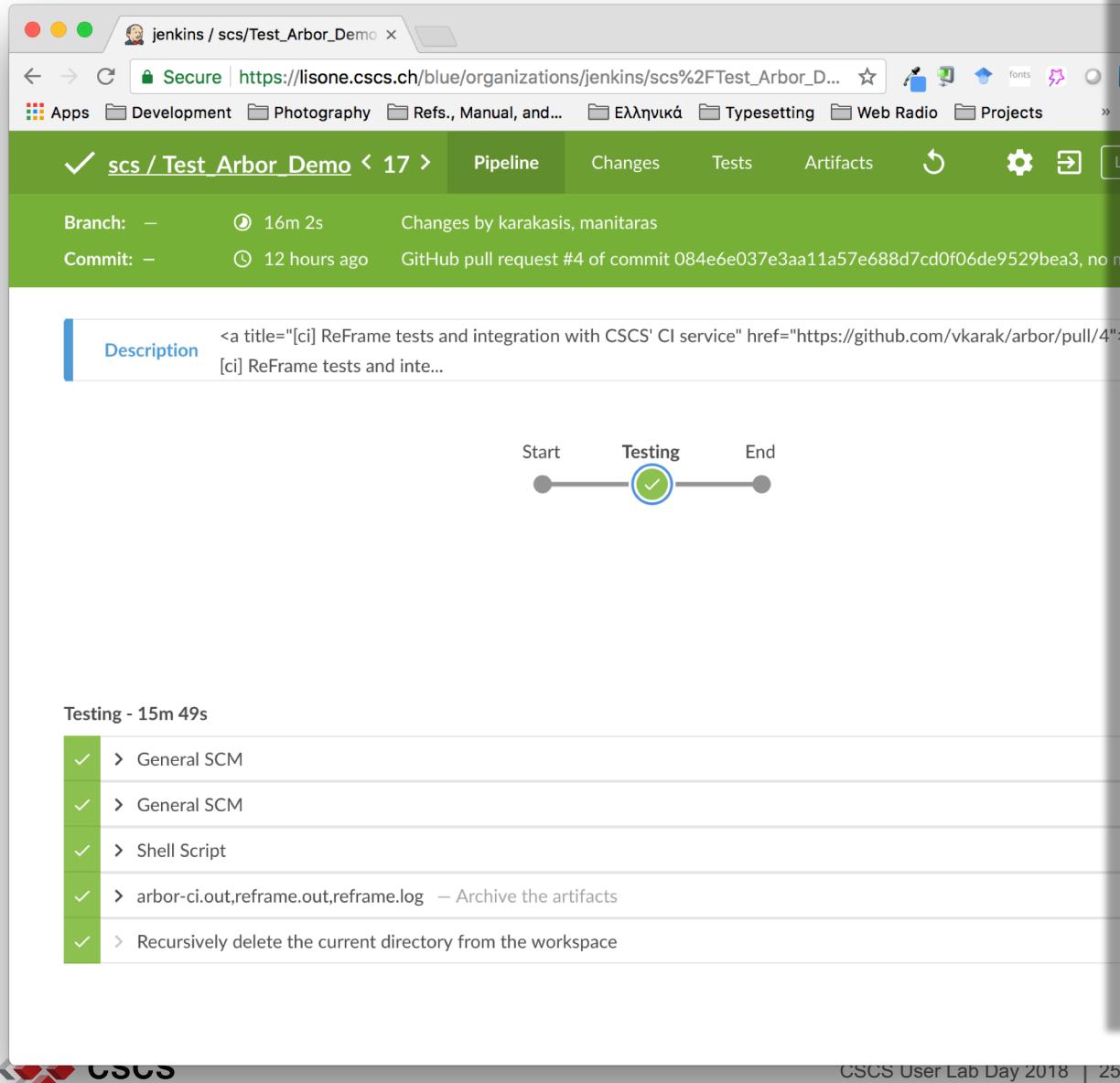
Open a Pull Request

- As soon as you open a PR or push to the PR, CSCS' CI will be triggered.



Demo PR: <https://github.com/vkarak/arbor/pull/4>

Examine ReFrame's Output Inside Jenkins



The screenshot shows a Jenkins pipeline job named "scs / Test Arbor Demo < 17 >". The pipeline has three stages: "Changes", "Tests", and "Artifacts". The "Tests" stage is currently running, indicated by a green circle with a checkmark labeled "Testing". The status bar at the bottom shows "Testing - 15m 49s". A sidebar on the left lists several tasks:

- > General SCM
- > General SCM
- > Shell Script
- > arbor-ci.out,reframe.out,reframe.log — Archive the artifacts
- > Recursively delete the current directory from the workspace

```
10  Reframe version: 2.14-dev2
11  Launched by user: jenscs
12  Launched on host: nid02200
13  Reframe paths
14  =====
15      Check prefix      :
16      Check search path : 'ci/arbor_tests.py'
17      Stage dir prefix   : /scratch/sn3000/jenscs/arbor-ci-084e6e0-17/stage/
18      Output dir prefix   : /scratch/sn3000/jenscs/arbor-ci-084e6e0-17/output/
19      Perf. logging prefix : /scratch/sn3000/jenscs/arbor-ci-084e6e0-17/perflogs
20  [=====] Running 6 check(s)
21  [=====] Started on Fri Sep 7 10:36:41 2018
22
23  [-----] started processing ArborBaseTest (ArborBaseTest)
24  [ RUN    ] ArborBaseTest on daint:gpu using PrgEnv-gnu
25  [-----] finished processing ArborBaseTest (ArborBaseTest)
26
27  [-----] started processing ArborMPITest (ArborMPITest)
28  [ RUN    ] ArborMPITest on daint:gpu using PrgEnv-gnu
29  [-----] finished processing ArborMPITest (ArborMPITest)
30
31  [-----] started processing ArborGpuTest (ArborGpuTest)
32  [ RUN    ] ArborGpuTest on daint:gpu using PrgEnv-gnu
33  [-----] finished processing ArborGpuTest (ArborGpuTest)
34
35  [-----] started processing ArborSIMDTest_haswell (ArborSIMDTest_haswell)
36  [ RUN    ] ArborSIMDTest_haswell on daint:gpu using PrgEnv-gnu
37  [-----] finished processing ArborSIMDTest_haswell (ArborSIMDTest_haswell)
38
39  [-----] started processing ArborSIMDTest_broadwell (ArborSIMDTest_broadwell)
40  [-----] finished processing ArborSIMDTest_broadwell (ArborSIMDTest_broadwell)
41
42  [-----] started processing ArborSIMDTest_native (ArborSIMDTest_native)
43  [-----] finished processing ArborSIMDTest_native (ArborSIMDTest_native)
44
45  [-----] waiting for spawned checks to finish
46  [   OK  ] ArborGpuTest on daint:gpu using PrgEnv-gnu
47  [   OK  ] ArborBaseTest on daint:gpu using PrgEnv-gnu
48  [   OK  ] ArborMPITest on daint:gpu using PrgEnv-gnu
49  [   OK  ] ArborSIMDTest_haswell on daint:gpu using PrgEnv-gnu
50  [-----] all spawned checks have finished
51
52  [  PASSED  ] Ran 4 test case(s) from 6 check(s) (0 failure(s))
53  [=====] Finished on Fri Sep 7 10:50:08 2018
```

✓ > arbor-ci.out,reframe.out,reframe.log — Archive the artifacts
✓ > Recursively delete the current directory from the workspace

Conclusions and Future Directions

ReFrame is a powerful tool that allows you to leverage continuous integration in an HPC environment without having to deal with the low-level system interaction details.

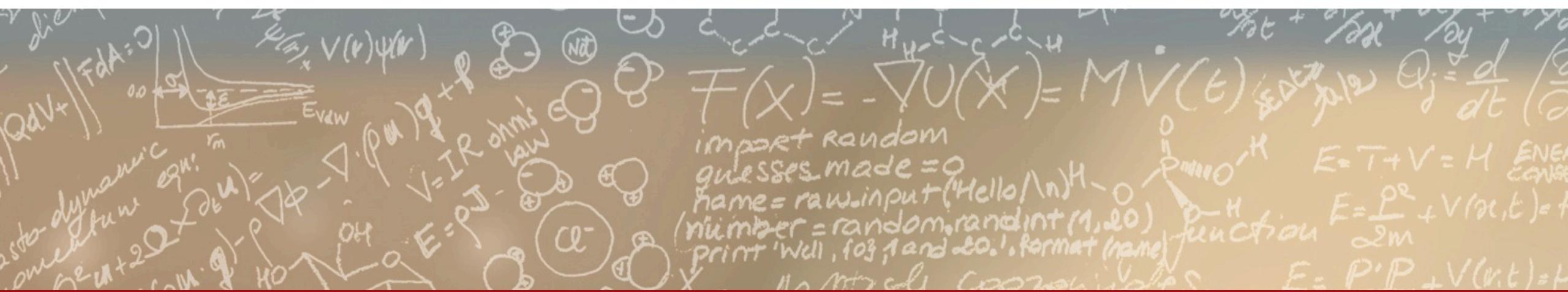
- High-level tests written in Python
- Portability across HPC system platforms
- Comprehensive reports and reproducible methods
- ReFrame is being actively developed with a monthly release-cycle.
- Future directions
 - Full integration with CSCS' CI service (no need for the intermediate submission script)
 - Test dependencies
 - Support for containers
 - Frontend and API improvements
- Bug reports, feature requests, help @ <https://github.com/eth-cscs/reframe>



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.