ReFrame
A Tool for Enabling Regression Testing and Continuous Integration for HPC Systems

HPC Knowledge Meeting '19
June 13th, Barcelona
Vasileios Karakasis, CSCS

reframe@cscs.
https://reframe-hpc.readthedocs.io
https://github.com/eth-cscs/reframe
https://reframe-slack.herokuapp.com
Why Regression Testing?

- The HPC scientific software stack is highly complex and very sensitive to changes.
- How can we ensure that the user experience is unaffected after an upgrade or after an “innocent” change in the system configuration?
- How testing of such complex systems can be made sustainable?
  - Consistency
  - Maintainability
  - Automation
Background

- CSCS had a shell-script based regression testing suite
  - Tests very tightly coupled to system details
  - Lots of code replication across tests
  - 15K lines of test code

- Simple changes required significant team effort
  - Porting all tests to native Slurm took several weeks

- Fixing even simple bugs was a tedious task
  - Tens of regression test files had to be fixed
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.

https://github.com/eth-cscs/reframe
Timeline / ReFrame Evolution

Regression testing framework starts as a pilot project

PyRegression 2.0

Production

First public release

ReFrame 2.4

Development moves to Github

ReFrame 2.18

03/16

12/16

04/17

02/18

06/19

5x reduction in tests code; more coverage

Asynchronous execution of tests

CSCS checks published

26 forks

49 stargazers

> 10 external users
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 Syslog and Graylog
- Clean internal APIs that allow the easy extension of the framework’s functionality
More Features

- Multiple workload manager backends
  - SLURM
  - PBS/Torque
- Multiple parallel launcher backends
  - srun, mpirun, mpiexec etc.
- Multiple environment modules backends
  - Tmod, Tmod4, Lmod
- Build system backends
  - CMake, Autotools, Make
- Asynchronous execution of regression tests
- Complete documentation (tutorials, reference guide)
- ... and more (https://github.com/eth-cscs/reframe)
ReFrame’s architecture

Developer of regression tests
@rfm.simple_test
class MyTest(rfm.RegressionTest):

ReFrame Frontend
reframe -r

Regression Test API
System abstractions
Job schedulers
Job launchers
Build systems
Environment modules
Environment abstractions
Operating System
Pluggable backends
site_configuration = {
    'systems': {
        'ault': {
            'descr': 'Ault TDS',
            'hostnames': ['ault'],
            'modules_system': 'lmod',
            'resourcesdir': '/apps/common/UES/reframe/resources',
            'partitions': {
                'login': {
                    'scheduler': 'local',
                    'environs': ['PrgEnv-gnu'],
                    'descr': 'Login nodes',
                    'max_jobs': 4
                },
                'amdv100': {
                    'scheduler': 'nativeslurm',
                    'access': ['-pamdv100'],
                    'environs': ['PrgEnv-gnu'],
                    'descr': 'Intel Skylake 36c + 4x NVIDIA V100',
                    'max_jobs': 100
                }
            }
        },
        'amdv100': {
            'scheduler': 'nativeslurm',
            'access': ['-pamdv100'],
            'environs': ['PrgEnv-gnu'],
            'descr': 'Intel Skylake 36c + 4x NVIDIA V100',
            'max_jobs': 100
        }
    }"
import reframe as rfm
import reframe.utility.sanity as sn

@rfm.simple_test
class Example7Test(rfm.RegressionTest):
def __init__(self):
    super().__init__()
    self.descr = 'Matrix-vector multiplication (CUDA performance test)'
    self.valid_systems = ['daint:gpu']
    self.valid_prog_environs = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
    self.sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
    self.build_system = 'SingleSource'
    self.build_system.cxxflags = ['-O3']
    self.executable_opts = ['4096', '1000']
    self.modules = ['cudatoolkit']
    self.num_gpus_per_node = 1
    self.sanity_patterns = sn.assert_found(r'time for single matrix vector multiplication', self.stdout)
    self.perf_patterns = {
        'perf': sn.extraxtsingle(r'^Performance:.*Performance:\s+\(<Gflops\>\s+)\d+.*\)',
        self.stdout, 'Gflops', float)
    }
    self.reference = {
        'daint:gpu': {
            'perf': (50.0, -0.1, 0.1, 'Gflop/s'),
        }
    }
    self.maintainers = ['you-can-type-your-email-here']
    self.tags = {'tutorial'}
The Regression Test Pipeline / How ReFrame Executes Tests

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

Serial execution policy

Asynchronous execution policy
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
The Regression Test Pipeline / How ReFrame Handles the Environment

ReFrame ...

- does not rely on module purge
  - *This is a direct road to disaster for Cray systems*
- starts in the unmodified user environment
- guarantees that each test case will run in the environment that ReFrame was originally invoked in
  - Saves the current environment before entering the setup stage of a test case
  - Restores it when exiting the cleanup stage of a test case
- resolves automatically module conflicts and generates the correct module load/unload sequence
  - Generates build and/or run scripts that can be used to reproduce the framework’s behavior
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 Jun  7 17:50:58 2019

[--------] 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 Jun  7 17:51:07 2019

==============================================================================
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: performance error: failed to meet reference: perf=50.358136, expected 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 formatting is fully configurable

- ReFrame can also send logs to Syslog or directly to a Graylog server.
Running ReFrame / Benchmarking mode

- Run with `--performance-report` option

```
PERFORMANCE REPORT
------------------------------------------------------------------------------
KernelLatencyTest_sync
- dom:gpu
  - PrgEnv-cray
    * latency: 6.57793 us
  - PrgEnv-gnu
    * latency: 6.39101 us
------------------------------------------------------------------------------
KernelLatencyTest_async
- dom:gpu
  - PrgEnv-cray
    * latency: 2.33462 us
  - PrgEnv-gnu
    * latency: 2.36944 us
```

```
self.perf_patterns = {
    'latency': sn.max(sn.extractall(
        r'\[\S+\] \[gpu \d+\] Kernel launch latency: ' 
        r'(?P<latency>[\S ]+) us', self.stdout, 'latency', float))
}

# Specify references per kernel version (sync/async)
self.sys_reference = {
    'sync': {
        'dom:gpu': {'latency': (6.6, None, 0.10, 'us')},
        'daint:gpu': {'latency': (6.6, None, 0.10, 'us')},
        'kesch:cn': {'latency': (12.0, None, 0.10, 'us')}
    },
    'async': {
        'dom:gpu': {'latency': (2.2, None, 0.10, 'us')},
        'daint:gpu': {'latency': (2.2, None, 0.10, 'us')},
        'kesch:cn': {'latency': (5.7, None, 0.10, 'us')}
    }
}
self.reference = self.sys_reference[kernel_version]
```
Running ReFrame / Flexible tests

- Setting num_tasks to zero will cause the test to be spawned on any number of nodes.

- Three modes of execution
  1. --flex-alloc-tasks=idle  Test will be spawned on all idle nodes (default)
  2. --flex-alloc-tasks=all  Test will be spawned on all nodes of the virtual partition
  3. --flex-alloc-tasks=<NUM>  Test will use NUM nodes

- May be combined with any other option restricting the set of nodes:
  - --flex-alloc-tasks=all --reservation=syscheckout
  - --flex-alloc-tasks=all --nodelist=nid00[0001-0010]
  - --flex-alloc-tasks=all --partition=foo --constraint=bar

- Valid only for the Slurm backend
Using ReFrame at CSCS
ReFrame @ CSCS / Tests

- Used for continuously testing systems in production
  - Piz Daint: 332 tests
  - Piz Kesch: 85 tests
  - Leone: 48 tests
  - Total: 542 different tests (reused across systems)

- Three categories of tests
  1. Production
     - Applications, libraries, programming environments, profiling tools, debuggers, microbenchmarks
     - Sanity and performance
     - Run nightly by Jenkins
  2. Maintenance
     - Programming environment sanity and key user applications performance
     - Before/after maintenance sessions
  3. Diagnostics and benchmarks
ReFrame @ CSCS / Production set-up
ReFrame @ CSCS / Production set-up
Using ReFrame with a CI service
ReFrame integration with CI services

- ReFrame is **NOT** a unit test framework

- ReFrame does **NOT** replace CI/CD frameworks

- It helps with integration, functional and performance tests on **HPC** systems and provides a uniform way of writing your tests across systems.
ReFrame integration with CI services

- CSCS CI service
  - Based on Jenkins
  - Run on CSCS HPC systems
  - On the remote side there is a Jenkins VM that can only run `sbatch` to the compute nodes
  - Integration steps
    1. Add a Jenkinsfile to project
    2. Add a batch script for running ReFrame on the compute nodes
    3. Add configuration entry for the target systems
    4. Add ReFrame tests

- Travis – Github
  - Runs on a VM in the cloud
  - Integration steps
    1. Clone ReFrame as well in your `.travis.yml`
    2. Add configuration entry for the Travis VM
    3. Add ReFrame tests
ReFrame integration with CI services / Arbor example

```python
@rfm.simple_test
class ArborBaseTest(rfm.RegressionTest):
    def __init__(self):
        super().__init__()

    @rfm.parameterized_test([['haswell'], ['broadwell'], ['native']])
    class ArborSIMDTest(ArborBaseTest):
        def __init__(self, arch_kind):
            super().__init__()
            if arch_kind == 'haswell':
                self.valid_systems = ['daint:gpu']
            elif arch_kind == 'broadwell':
                self.valid_systems = ['daint:mc', 'tresa']
            elif arch_kind == 'native':
                self.valid_systems = ['tresa']

            self.arch_kind = arch_kind
            self.build_system.config_opts += ["-DARB_VECTORIZE=ON",
                                           '-DARB_ARCH=%s' % self.arch_kind]
            self.tags = ("sanity")
```

You can use inheritance to avoid redefining common functionality!

Use parameterized tests to create test factories!
ReFrame with CSCS CI service
Upcoming Features / Test Dependencies

```python
@rfm.simple_test
class BuildMyAppTest(rfm.CompileOnlyRegressionTest):
    """A test that requires a long compilation"""
    def __init__(self):
        super().__init__()
        ...

@rfm.parameterized_test(*([2**n] for n in range(11)))
class ScalingTest(rfm.RunOnlyRegressionTest):
    def __init__(self, num_tasks):
        super().__init__()
        self.num_tasks = num_tasks
        self.depends_on('BuildMyAppTest')
        ...

    def setup(self, *args, **kwargs):
        target = self.getdep('BuildMyAppTest')
        self.executable = os.path.join(target.stagedir, 'bin', 'myapp.exe')
        self.executable_opts = ['arg0', 'arg1']
        super().setup(*args, **kwargs)
```
Upcoming Features / Seamless Container Support

```python
@rfm.simple_test
class MyAppTest(rfm.RunOnlyRegressionTest):
    def __init__(self):
        super().__init__(self)
        self.num_tasks = 16
        self.container_platform = 'Singularity'  # or 'Docker', 'Shifter' etc.
        self.container_platform.image = 'ubuntu:tag'
        self.container_platform.mount_points = [
            ('/path/to/my/openmpi', '/opt/local/openmpi')
        ]
        self.container_platform.commands = [
            'export LD_LIBRARY_PATH=/opt/local/openmpi/lib:$LD_LIBRARY_PATH',
            './bin/run_my_exec'
        ]
...
```
Who is using or experimenting with ReFrame?
Conclusions

ReFrame is a powerful tool that allows you to continuously test 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
- Functionality to test multiple nodes at the same time

- Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe
Thank you for your attention.

reframe@cscs.ch
https://eth-cscs.github.io/reframe
https://github.com/eth-cscs/reframe
https://reframe-slack.herokuapp.com
Acknowledgements

- **Framework contributions**
  - Andreas Jocksch, CSCS
  - Christopher Bignamini, CSCS
  - Matthias Kraushaar, CSCS
  - Rafael Sarmiento, CSCS
  - Samuel Omlin, CSCS
  - Theofilos Manitaras, CSCS
  - Vasileios Karakasis, CSCS
  - Victor Holanda, CSCS

- **Regression tests**
  - SCS and OPS team
ReFrame with Travis
Why ReFrame?

- As a sysadmin
  - Maintainability
  - Portability
  - Ease of use
  - Efficiency
  - Logging

- As a scientific software developer
  - Harness HPC systems
  - Reuse tests across multiple systems
  - Increase productivity
Other HPC regression testing tools

- **JUBE**
  - Focused on benchmarking
  - Tests written in XML

- **Buildtest**
  - Tied to EasyBuild installations
  - Tests written in YAML

- **Testpilot**
  - Collection of tools
  - Tests written in Bash; multiple scripts required per test
  - Mostly PBS