CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

ReFrame: A Framework for Writing Regression Tests for HPC Systems

5th EasyBuild User Meeting
Barcelona, Spain

Vasileios Karakasis, CSCS

January 30, 2020

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

Why regression testing?

- The HPC 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 and low coverage

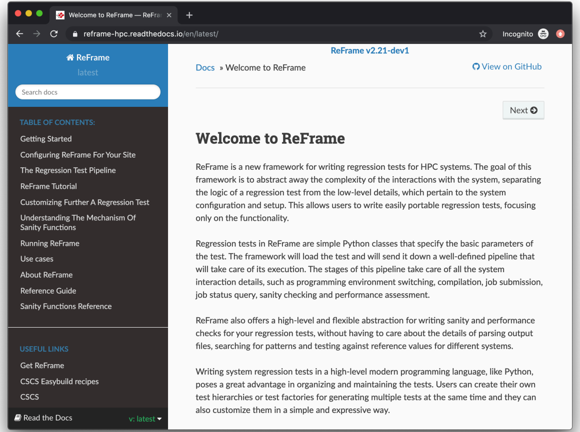
- Simple changes required significant team effort

- Fixing even simple bugs was a tedious task

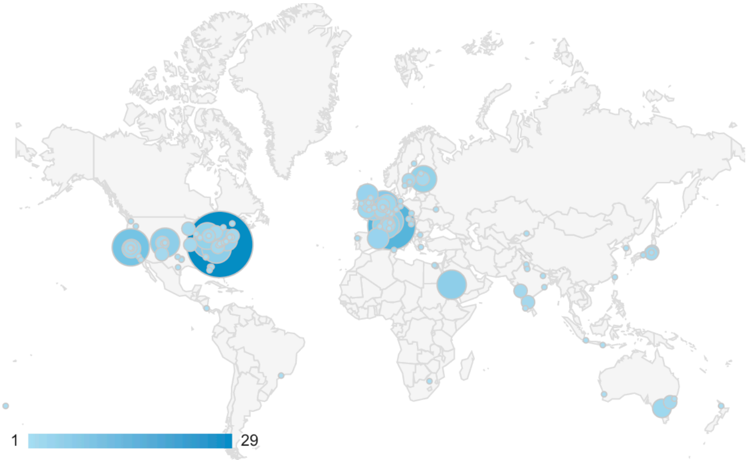
What is ReFrame?

An HPC 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,
- provides a runtime for running **efficiently** the regression tests.



Who is using ReFrame or is curious about it?



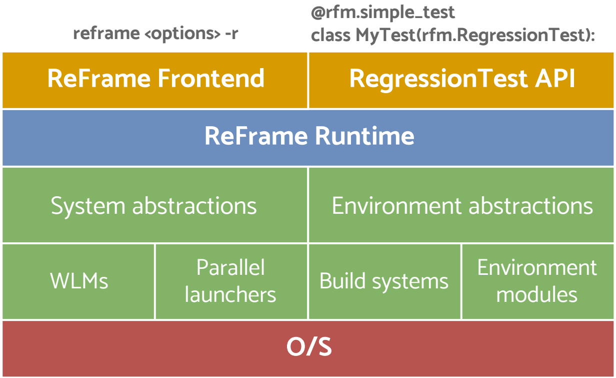
Design Goals

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

Key Features

- Support for cycling through programming environments and system partitions
- Support for different WLMs, parallel job launchers and modules systems
- Support for sanity and performance tests
- Support for test factories
- Support for container runtimes (new in v2.20)
- Support for test dependencies (new in v2.21)
- Concurrent execution of regression tests
- 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

ReFrame's Architecture



How ReFrame Executes the Tests

All tests go through a well-defined pipeline.

Setup	Build	Run	Sanity	Perf.	Cleanup

The regression test pipeline

How ReFrame Executes the Tests

All tests go through a well-defined pipeline.

The regression test pipeline

Serial execution policy

How ReFrame Executes the Tests

All tests go through a well-defined pipeline.

The regression test pipeline

Serial execution policy

Asynchronous execution policy

CSCS

ETH zürich

Configuring ReFrame

1. Systems
   - Hostname patterns that will let ReFrame recognize this system
   - Modules system used
   - Define system's virtual partitions
2. Virtual partitions
   - Job scheduler and parallel job launcher
   - How access to this partition is granted
   - The programming environments to be tested on this partition
3. Programming environments (toolchains)
   - Environment modules to load
   - Environment variables to set

https://github.com/eth-cscs/reframe/blob/master/config/cscs.py

Writing a Regression Test in ReFrame

```
import reframe as rfm
import reframe.utility.sanity as sn


@rfm.simple_test
class Example3Test(rfm.RegressionTest):
    def __init__(self):
        self.descr = 'Matrix-vector␣multiplication␣example␣with␣MPI+OpenMP'
        self.valid_systems = ['daint:gpu', 'daint:mc']
        self.valid_prog_environs = ['PrgEnv-cray', 'PrgEnv-gnu', 'PrgEnv-intel', 'PrgEnv-pgi']
        self.sourcepath = 'example_matrix_vector_multiplication_mpi_openmp.c'
        self.build_system = 'SingleSource'
        self.executable_opts = ['1024', '10']
        self.prgenv_flags = {'PrgEnv-cray':  ['-homp'],
                             'PrgEnv-gnu':   ['-fopenmp'],
                             'PrgEnv-intel': ['-openmp'],
                             'PrgEnv-pgi':   ['-mp']}
        self.sanity_patterns = sn.assert_found(r'time␣for␣single␣matrix␣vector␣multiplication', self.stdout)
        self.num_tasks = 8
        self.num_tasks_per_node = 2
        self.num_cpus_per_task = 4
        self.variables = {'OMP_NUM_THREADS': str(self.num_cpus_per_task)}
        self.tags = {'tutorial'}

    @rfm.run_before('compile')
    def setflags(self):
        self.build_system.cflags = self.prgenv_flags[self.current_environ.name]
```

CSCS

ETH zürich

Writing a Performance Test in ReFrame

```python
import reframe as rfm
import reframe.utility.sanity as sn


@rfm.simple_test
class Example7Test(rfm.RegressionTest):
    def __init__(self):
        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.sanity_patterns = sn.assert_found(r'time␣for␣single␣matrix␣vector␣multiplication', self.stdout)
      ⟶self.perf_patterns = {
            'perf': sn.extractsingle(r'Performance:\s+(?P<Gflops>\S+)␣Gflop/s', self.stdout, 'Gflops', float)
        }
      ⟶self.reference = {
            'daint:gpu': {
                'perf': (50.0, -0.1, 0.1, 'Gflop/s'),
            }
        }
        self.tags = {'tutorial'}
```

Defining Test Dependencies

```python
class BaseTest(rfm.RunOnlyRegressionTest):
    def __init__(self):
        self.valid_systems = ['*']
        self.valid_prog_environs = ['*']
        self.sourcesdir = None
        self.executable = 'echo'
        self.count = sn.getattr(self, '_count')
        self.sanity_patterns = sn.defer(True)
        self.keep_files = ['out.txt']
        self._count = int(type(self).__name__[1:])

    @rfm.run_before('run')
    def write_count(self):
        self.executable_opts = [str(self.count),
                                '>␣out.txt']
```

```python
@rfm.simple_test
class T0(BaseTest):
    pass

@rfm.simple_test
class T4(BaseTest):
    def __init__(self):
        super().__init__()
        self.depends_on('T0')
        self.sanity_patterns = sn.assert_eq(self.count, 4)

    @rfm.require_deps
    def prepend_output(self, T0):
        with open(os.path.join(T0().stagedir, 'out.txt')) as fp:
            self._count += int(fp.read())
```

- Dependent tests can access all the resources of their parent tests
- Runtime takes care of the correct execution of the tests and the cleanup of their resources
- Dependencies can be defined at the level of programming environment as well

Running ReFrame

Sample output with the asynchronous execution policy

```
[==========] Running 1 check(s)
[==========] Started on Sat Nov 16 20:33:11 2019

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

[----------] waiting for spawned checks to finish
[       OK ] Example7Test on daint:gpu using PrgEnv-cray
[       OK ] Example7Test on daint:gpu using PrgEnv-gnu
[       OK ] Example7Test on daint:gpu using PrgEnv-pgi
[----------] all spawned checks have finished

[  PASSED  ] Ran 3 test case(s) from 1 check(s) (0 failure(s))
[==========] Finished on Sat Nov 16 20:33:25 2019
```

Running ReFrame

Sample failure

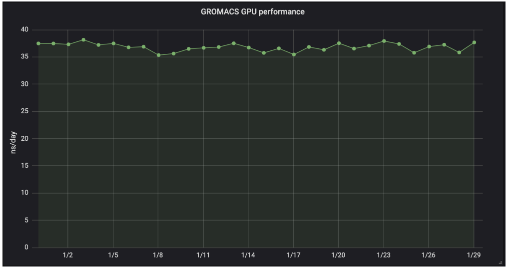
```
[==========] Running 1 check(s)
[==========] Started on Thu Jan 30 00:34:17 2020

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

[----------] waiting for spawned checks to finish
[     FAIL ] Example7Test on daint:gpu using PrgEnv-gnu
[----------] all spawned checks have finished

[  FAILED  ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[==========] Finished on Thu Jan 30 00:34:25 2020

==============================================================================
SUMMARY OF FAILURES
------------------------------------------------------------------------------
FAILURE INFO for Example7Test
  * System partition: daint:gpu
  * Environment: PrgEnv-gnu
  * Stage directory: /users/karakasv/Devel/reframe/stage/daint/gpu/PrgEnv-gnu/Example7Test
  * Node list: nid00000
  * Job type: batch job (id=905395)
  * Maintainers: ['you-can-type-your-email-here']
  * Failing phase: performance
  * Reason: performance error: failed to meet reference: perf=50.050688, expected 70.0 (l=63.0, u=77.0)
------------------------------------------------------------------------------
```

Running ReFrame

Performance logging

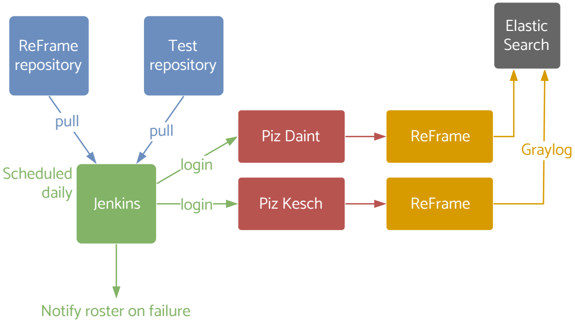
- Every time a performance test is run, ReFrame can log its performance through several channels:
  - Normal files
  - Syslog
  - Graylog

- Log format is fully configurable



ReFrame @ CSCS

Tests and production setup

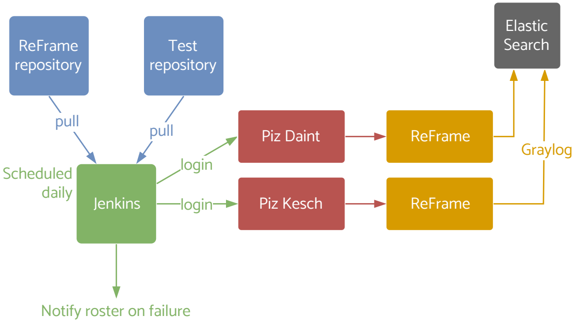


Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- 534 tests in total (90 test files)

ReFrame @ CSCS

Tests and production setup



Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- 534 tests in total (90 test files)

Experiences from Piz Daint's upgrade to CLE7:

- Enabling ReFrame as early as possible on the TDS has streamlined the upgrade process.
- Revealed several regressions in the programming environment that needed to be fixed.
- Builds confidence when finally everything is GREEN.

ReFrame @ CSCS

Test suite

- HPC applications: Amber, CP2K, CPMD, QuantumEspresso, GROMACS, LAMMPS, NAMD, OpenFoam, Paraview, TensorFlow
- Libraries: Boost, GridTools, HPX, HDF5, NetCDF, Magma, Scalapack, Trilinos, PETSc
- Programming environment: GPU, MPI, MPI+X functionality, OpenACC, CPU affinity
- Slurm functionality
- Performance and debugging tools
- I/O tests: IOR
- Microbenchmarks: CUDA, CPU, MPI
- Sarus container runtime checks
- OpenStack: S3 API

Check the “cscs-checks/” directory @ https://github.com/eth-cscs/reframe

ReFrame @ Other Sites

- National Energy Research Scientific Computing Center, USA
  - Software stack validation
  - Performance testing and benchmarking
  - Integration with Gitlab CI/CD solution developed within ECP
  - V. Karakasis et al., “Enabling Continuous Testing of HPC Systems using ReFrame”, HUST’19
- Ohio Supercomputing Center, USA
  - Software stack validation
  - Integration with CI/CD
  - S. Khuvis et al., “A Continuous Integration-Based Framework for Software Management”, PEARC’19
- PAWSEY (AUS), NIWA (NZ), SurfSARA (NL), ASML (NL) and many more experimenting

Using ReFrame to Test EasyBuild (work-in-progress)

- A ReFrame test for each easyconfig file that will run EasyBuild to install it and check for successful completion

Using ReFrame to Test EasyBuild (work-in-progress)

- A ReFrame test for each easyconfig file that will run EasyBuild to install it and check for successful completion
- The dependency graph is generated on-the-fly by calling the EasyBuild API
- Use a parameterized ReFrame test with the dependency information and let ReFrame generate all the easyconfig tests at once!

```
# Call EasyBuild API to determine easyconfig deps and generate ec_tests
#  - Each element of ec_tests is a tuple of  [name, ec['spec'], test_deps]
@rfm.parameterized_test(*ec_tests)
class EasyconfigTest(rfm.RunOnlyRegressionTest):
    def __init__(self, name, ec_file, deps):
        self.name = name
        self.executable = 'eb'
        self.executable_opts = [ec_file, '--force', '--module-only']
        for dep in deps:
            self.depends_on(dep)
        # ...
```

Full code snippet: https://gist.github.com/boegel/22defbfae0bcc7a9b0b76d8e40040f94

ReFrame Roadmap for 2020

- Redesign the configuration component
  - New configuration syntax with more control on the different aspects of the framework
  - Multiple configuration formats (JSON, YAML, Python)
  - Enable configuration through environment variables
- Improve documentation
  - Targeted tutorials for EasyBuild/Spack installations and Cray systems
  - Advanced topics on writing tests
- Investigate ways of further facilitating the porting of tests to different systems
- Bug fixes and user feature requests
- ReFrame 3.0
  - https://github.com/eth-cscs/reframe/projects/15
  - Regular development releases

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
- Easy integration with CI/CD workflows

– Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe
– Sharing tests @ https://github.com/reframe-hpc

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Thank you for your attention

reframe@cscs.ch

https://reframe-hpc.readthedocs.io

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

https://reframe-slack.herokuapp.com