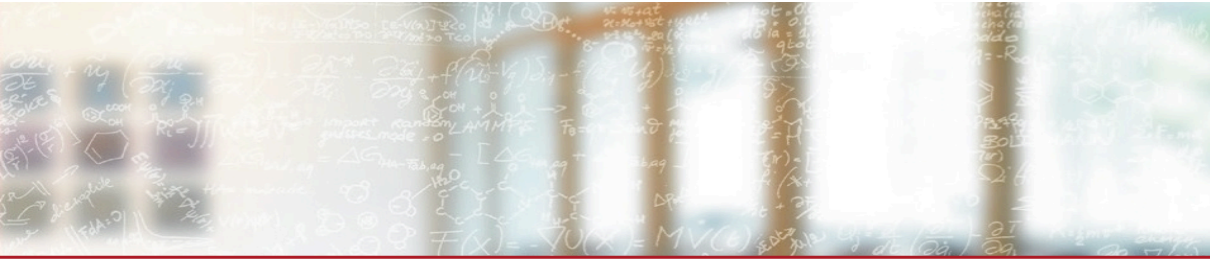
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Enabling Continuous Testing of HPC Systems using ReFrame

Sixth Annual Workshop on HPC User Support Tools (HUST 2019)
SC'19, Denver, CO, USA

Vasileios Karakasis, CSCS
Brian Friesen, NERSC
Zhi-Qiang You, OSC

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

November 18, 2019

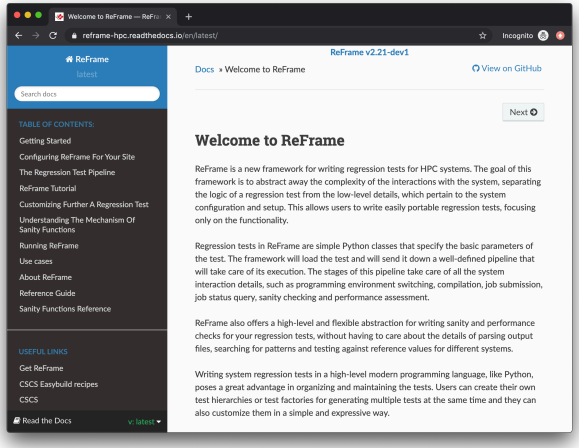
Why regression testing?

- 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.

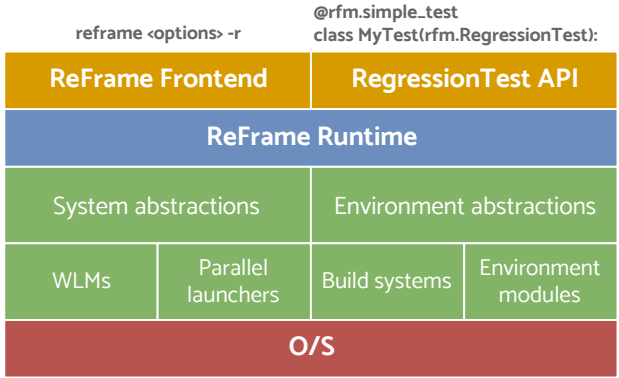
Design Goals

- 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.

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

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]
```

Writing a Performance Test in ReFrame

```
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'}
```

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
```

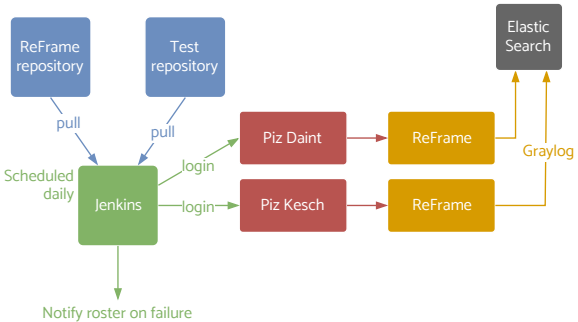
Running ReFrame

Sample failure

```
[==========] Running 1 check(s)
[==========] Started on 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)
------------------------------------------------------------------------------
```

ReFrame @ CSCS

Tests and production setup

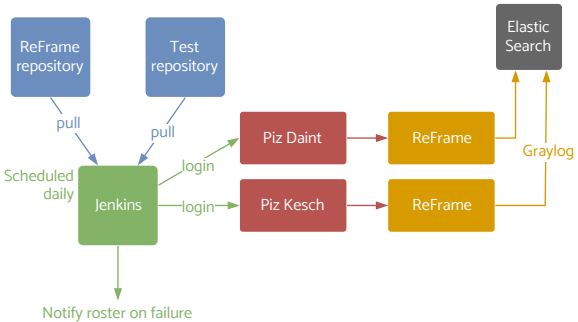


Several test categories identified by tags:

- 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 (most of them available on ReFrame's Github repo)

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 (most of them available on ReFrame's Github repo)

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.

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 in CI/CD workflows

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

ReFrame at NERSC

Brian Friesen, Helen He, Lisa Gerhardt, Brandon Cook, Christopher Samuel

National Energy Research Scientific Computing Center

Lawrence Berkeley National Laboratory

HUST'19

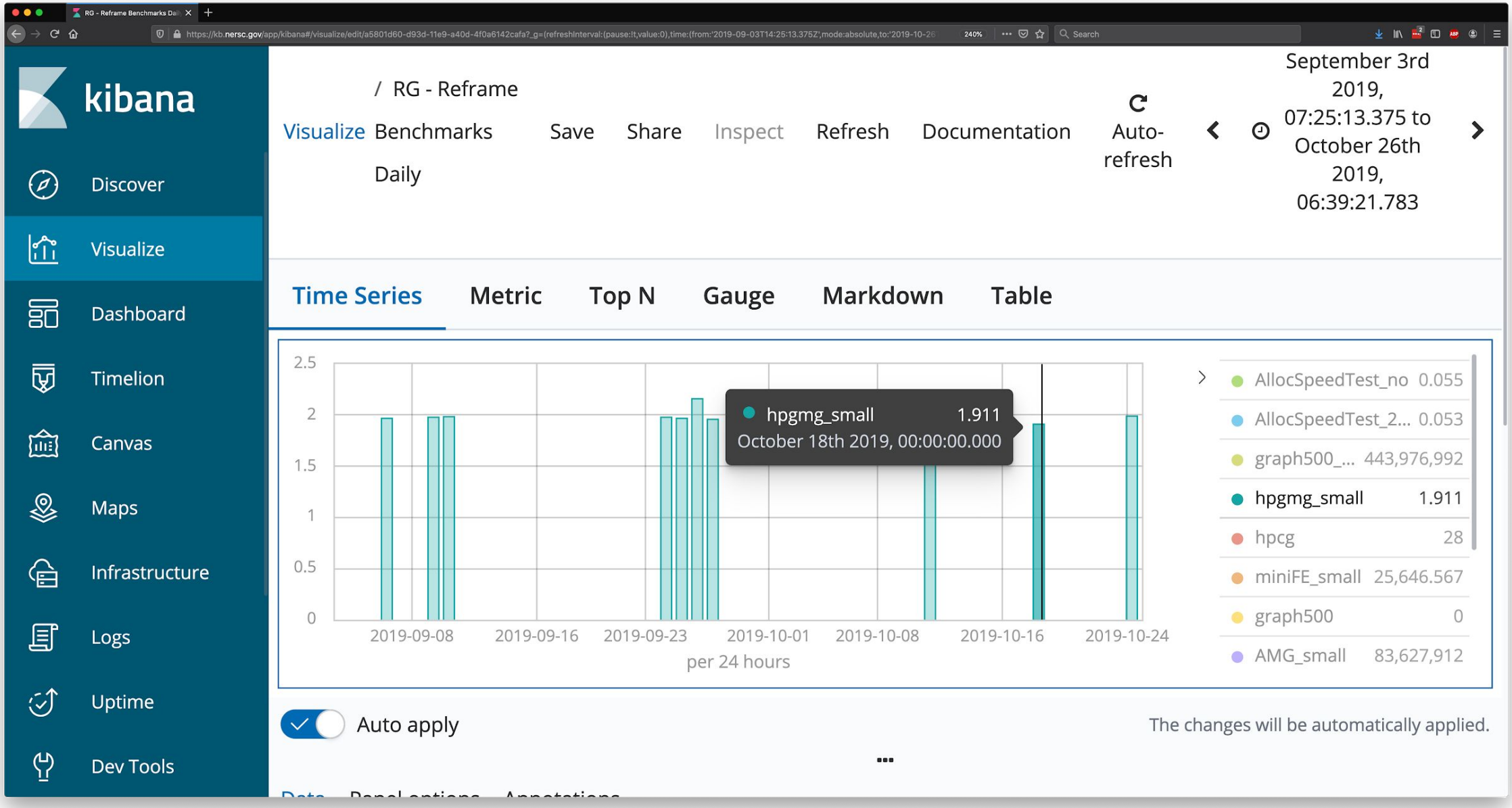
2019 Nov 18

System regression testing use cases at NERSC

- ‘Hotfixes’ applied to eLogins are often lost during reboots until the fix is is applied to node image
- Run as part of development work on Test & Dev System (TDS) during pre-maintenance stabilisation period to catch issues before deployment.
- System software changes/upgrades can be disruptive to large experiments and user facilities that use NERSC (DESI, ATLAS, LZ, ...)
  - can take weeks to reconfigure pipelines to address changes in system software locations, version changes, ABI changes
  - Experiments are contributing their own tests to NERSC’s ReFrame test battery to accelerate system verification for their own software
- Ongoing system performance monitoring

Integration with NERSC center-wide monitoring

- NERSC has consolidated system and facility monitoring data into a central Elasticsearch database
  - Integrated with Kibana and Grafana interfaces to create real-time web-based data dashboards, alerts
- Cray XC system monitoring data sent to Elasticsearch via Cray Lightweight Log Management (LLM) service
  - LLM listens on a syslog port
  - Collects data from hardware counters on compute nodes, cabinet temperature, power usage, etc., and now ReFrame performance test data too
- NERSC uses ReFrame's syslog logging interface to plug directly into LLM on Cori, such that all ReFrame performance tests are immediately logged in Elasticsearch
  - Data available on Kibana dashboard in real time


RG - Reframe Benchmarks Daily
https://kb.nersc.gov/app/kibana#/visualize/edit/a5801d60-d93d-11e9-a40d-4f0a6142cafa?_g=(refreshInterval:(pause:!t,value:0),time:(from:'2019-09-03T14:25:13.375Z',mode:absolute,to:'2019-10-26
240%
Search
kibana
Discover
Visualize
Dashboard
Timelion
Canvas
Maps
Infrastructure
Logs
Uptime
Dev Tools
Visualize / RG - Reframe Benchmarks Daily
Save
Share
Inspect
Refresh
Documentation
Auto-refresh
September 3rd 2019, 07:25:13.375 to October 26th 2019, 06:39:21.783
Time Series
Metric
Top N
Gauge
Markdown
Table
2.5
2
1.5
1
0.5
0
2019-09-08
2019-09-16
2019-09-23
2019-10-01
2019-10-08
2019-10-16
2019-10-24
per 24 hours
hpgmg_small 1.911
October 18th 2019, 00:00:00.000
AllocSpeedTest_no 0.055
AllocSpeedTest_2... 0.053
graph500_... 443,976,992
hpgmg_small 1.911
hpcg 28
miniFE_small 25,646.567
graph500 0
AMG_small 83,627,912
Auto apply
The changes will be automatically applied.

Test examples on Cori

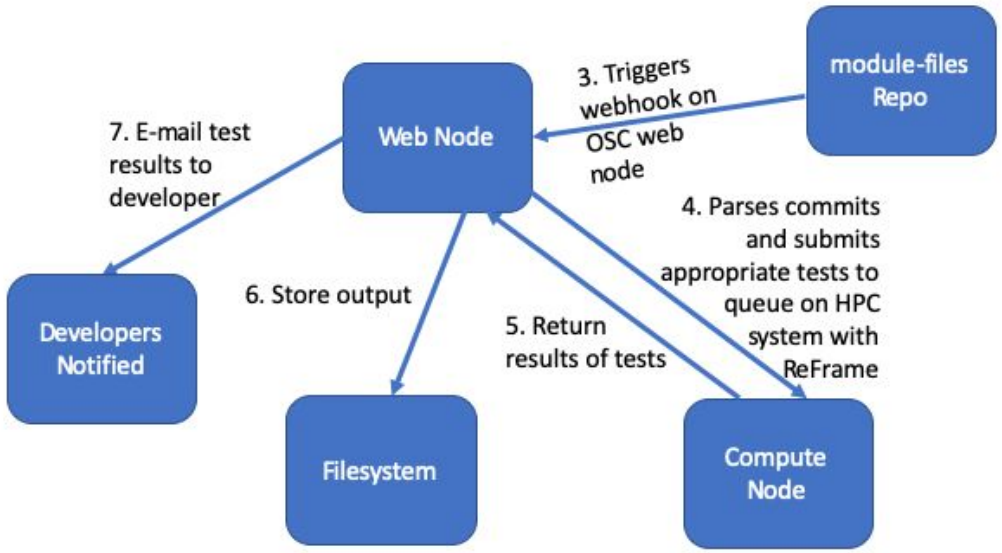
Functionality

- DataWarp stage in/out
- Shifter (pull/execute)
- Jupyter
- IDL, Matlab
- TensorFlow/PyTorch
- Dynamic RDMA credentials
- hugepage allocation
- HPSS
- (many others)

Performance

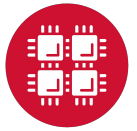
- NERSC-8 procurement benchmarks
- IOR
- HPGMG, Graph500, HPCG
- NESAP apps
- (several others)

Automated Testing



- After a software build completes, module is installed and committed to repository
- Repository is configured with a webhook that triggers appropriate tests on commit
- ReFrame is used to build testing system for software environment

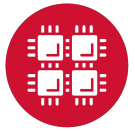
"A Continuous Integration-Based Framework for Software Management" PEARC' 19



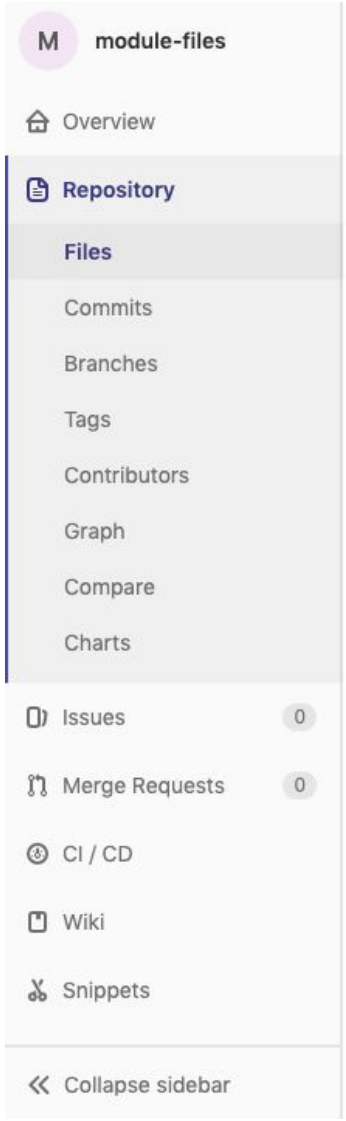
Application Testing

- ReFrame tests are performed by a user-privilege account

```
$ run-osc_regression.sh fftw3 3.3.8 mpi mvapich2 2.3 intel 18.0.3

Command line: /usr/local/reframe/2.17/reframe.py -C /path/to/reframe/settings.py -c
/path/to/reframe/checks/ -R --max-retries 1 --exec-policy=async --save-log-files
--nocolor -t fftw3$ -t apps -t version|module|test|perf -p intel-mvapich2 -M fftw3:
intel/18.0.3 mvapich2/2.3 fftw3/3.3.8 -r
```

Application Status



Application Status

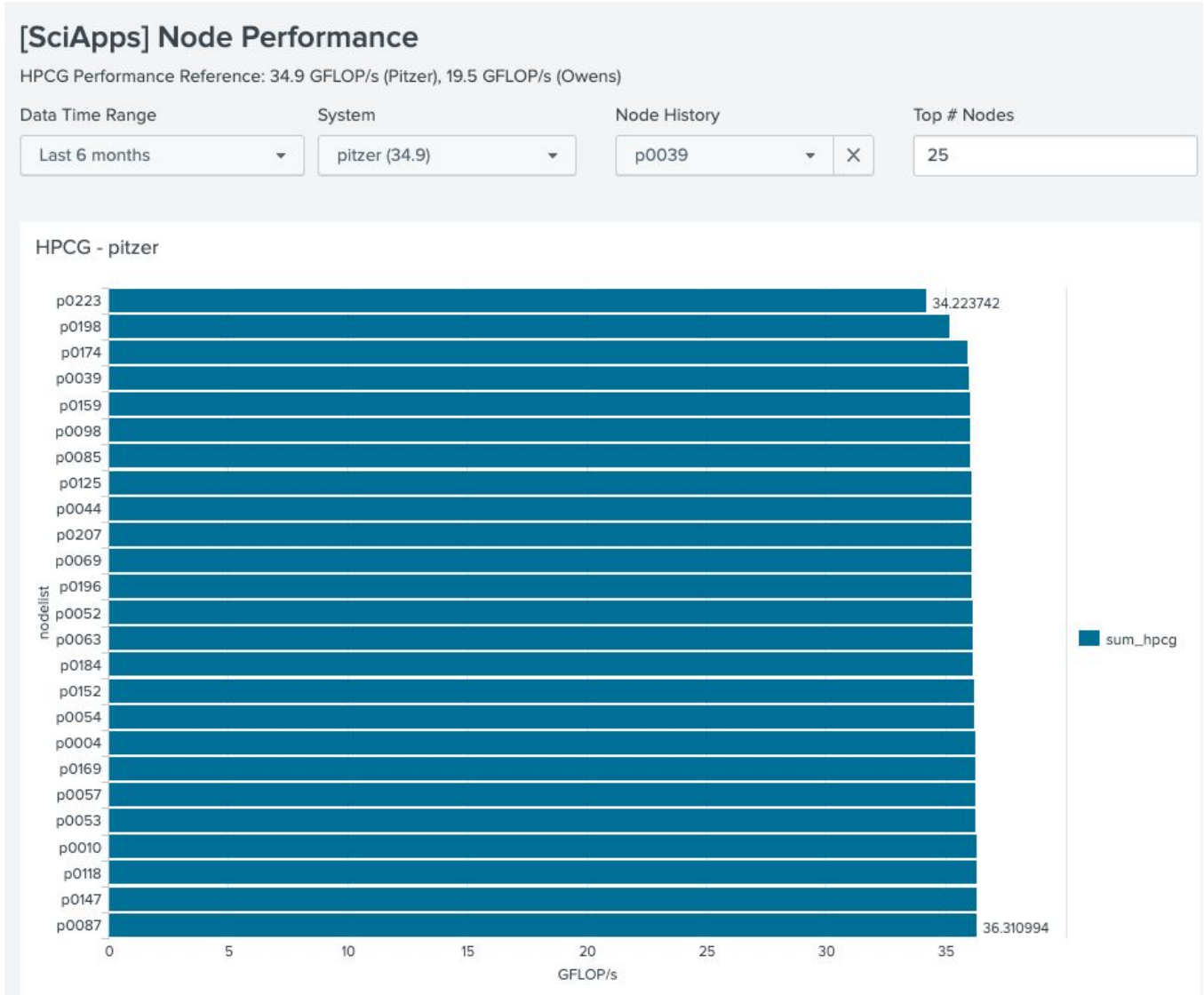
Application	Version	Dependencies	Status
R	3.5.0	compiler intel 18.0.3	✅
R	3.5.2	compiler intel 18.0.3	✅
ansys	19.2	core	✅
arm-ddt	19.0.1	core	✅
arm-map	19.0.1	core	✅
arm-pr	19.0.1	core	✅
boost	1.67.0	mpi mvapich2 2.3 intel 18.0.3	✅
cuda	10.0.130	core	✅
darshan	3.1.2	mpi mvapich2 2.2 intel 16.0.8	✅
darshan	3.1.2	mpi openmpi 1.10-hpcx intel 16.0.8	✅
darshan	3.1.2	mpi openmpi 1.10.7 intel 16.0.8	❌
darshan	3.1.4	mpi mvapich2 2.2 intel 16.0.8	✅
darshan	3.1.5	mpi mvapich2 2.2 intel 16.0.8	✅

System Regression Testing

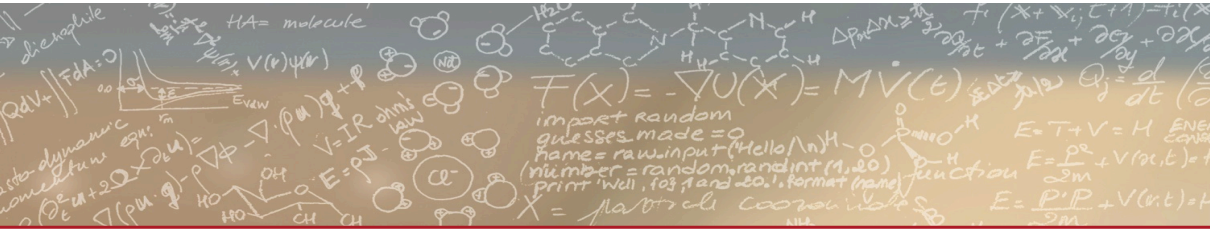
- Sanity check after system maintenance or updates to system software stack

```
$ run_reframe.sh -pid -t apps -r |& tee 20191008_downtime_gpfs.log
$ report_test.py 20191008_downtime_gpfs.log

Start Date: Tue Oct  8 17:42:27 2019
  End Date: Tue Oct  8 18:04:47 2019
Number of Checks: 254

ReFrame Test Summary

Total number of tests is 590
Total number of failures is 127

Phase        #      Comment
-----------  -----  ------------------------------------------------
setup        110    Can not load module or other unexpected errors
sanity       4      Sanity check failed; please rerun the tests:
run          0      Sanity check failed; please rerun the tests:
performance  3      Performance check failed; please rerun the tests:
....
```

Performance Monitoring



- Submit test once per month
- Record results in syslog format
- Upload to splunk, display in dashboard
- Can see performance variance in time and across nodes, similar to XDMoD capability



Thank you for your attention

reframe@cscs.ch

https://reframe-hpc.readthedocs.io

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

https://reframe-slack.herokuapp.com