Recent Advances in ReFrame

9th EasyBuild User Meeting

April 24, 2024

Vasileios Karakasis (NVIDIA)

ReFrame

ReFrame is a powerful framework that enables system testing and performance testing as code with unique HPC features.

- Composable tests written in Python allowing the creation of reusable test libraries
- Multi-dimensional test parameterisation
- Support for test fixtures
- Parallel execution of tests
- Programmable configuration
- Support for multiple HPC schedulers, modules systems, build systems and container runtimes
- Integration with Elastic and Graylog for feeding directly performance data from tests
- CI integration through Gitlab child pipelines

ReFrame community

- Documentation: https://reframe-hpc.readthedocs.io
  - >500 unique readers monthly from all over the world
- Slack workspace (289 members):
  - Join us through this link.
- Github
  - ReFrame HPC community group: https://github.com/reframe-hpc
    - Collection of public forks of site test repositories
  - 52 contributors since the beginning
  - Backlog: https://github.com/orgs/reframe-hpc/projects/1
  - Code: https://github.com/reframe-hpc/reframe
    - Give it a ⭐ !
- PyPI: https://pypi.org/project/ReFrame-HPC/
  - More than 10K downloads/month according to pepy.tech

New docs!

Large part of the documentation was rewritten from scratch (**v4.6**)

- New tutorial that covers all the modern aspects of ReFrame
  - Worth reading it even if you are an existing user!
  - https://reframe-hpc.readthedocs.io/en/stable/tutorial.html
- How To articles
  - https://reframe-hpc.readthedocs.io/en/stable/howto.html
- Fully containerized
  - Docker containers for single node tests
  - Docker compose with Slurm for multi-node tests

New features and enhancements

- Installation using the **`bootstrap.sh`** script
  - Support for multiple platforms (**v4.5**)
  - Support for pip-less environments (**v4.5**)
- Parallel launchers
  - Support for **`pdsh`** and **`clush`** (**v4.3**)
- Scheduler backends
  - New **`ssh`** backend (**v4.4**)
    - Copies test's artifacts to a remote host, launches the test and pulls back the generated artifacts
    - Only single-node jobs are supported
    - Multiple hosts can be specified and the **`--distribute`** option can be used to launch the same test on all remote hosts.
  - Support for writing custom scheduler backends outside the framework (**v4.4**)

New features and enhancements

- Environment configuration improvements
  - New **`prepare_cmds`** environment-level configuration option to emit commands before the environment is loaded (**v4.3**)
  - Skip sanity checking of a modules system backend, if module conflict resolution is off (**v4.5**)
    - Useful when the current system does not have a modules system installed, but its partitions have one
  - New **`nvcc`** option to set the NVIDIA CUDA compiler (**v4.6**)
- Spack integration improvements
  - New **`env_create_opts`** option to pass options to the **`spack env create`** command (**v4.3**)
  - New **`preinstall_cmds`** option to emit commands before spack install (**v4.3**)

New features and enhancements

- Expose more hardware info to the test
  - New **`model`** and **`vendor`** attributes in **`ProcessorInfo`** and **`DeviceInfo`** (**v4.6**)
    - E.g., `"Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz"`
  - New **`platform`** attribute **`ProcessorInfo`** (**v4.6**)
    - E.g., `"arm64"`, `"x86_64"`
- Flexible (Slurm) node allocation improvements (**v4.6**)
  - **`--distribute=STATE`** now distributes to nodes strictly in **`STATE`** state
    - E.g., **`--distribute=idle`** will not select nodes in **`IDLE+DRAIN`**
  - **`--distribute=STATE*`** will distribute to nodes that are at least in state **`STATE`**
  - New pseudo-state **`avail`** to select all nodes potentially available (nodes in any of **`ALLOCATED`**, **`IDLE`**, **`COMPLETING`** states)
  - Same rules apply for **`--flex-alloc-nodes`**

New features and enhancements

- Logging and performance logging
  - Job submit time is now logged through the **`check_job_submit_time`** placeholder (**v4.5**)
  - New placeholder **%(check_#ALL)s** that dumps all the loggable variables and parameters (**v4.3**)
  - Default perflog format has changed (**v4.3**)
  - Perflogs for parameterized are combined (**v4.3**)
  - Parameters and variables are loggable by default (**v4.5**)
- Test parameters and variables
  - Tests parameterization from the command-line with **`-P var=<values>`** (**v4.3**)
    - E.g., **`-P num_nodes=1,2,4,8,16`**
    - **`num_nodes`** must be defined as a variable (ReFrame will take care of any implicit conversion)
  - Nested mapping variables can be set with **`-S x='{"key0": {"key1": 1234}}'`** (**v4.3**)

New features and enhancements

- Support for custom system auto-detection methods (**v4.3**)
  - New top-level configuration option **`autodetect_methods`** or the **`RFM_AUTODETECT_METHODS`** env. variable
  - A list of auto-detection methods that generate a key that will be matched with the systems' hostnames
  - Can be arbitrary Python functions or external scripts
  - Very useful in cases where a system configuration cannot be derived from the hostname (e.g., cloud environments)

```
def get_vm_type():
    return
requests.get('http://169.254.169.254/latest/meta-data/i
nstance-type')

site_configuration = {
    'autodetect_methods': ['py::get_vm_type'],
    'systems': [
        {
            'name': 'm5n-large',
            'hostnames': ['m5n.large'],
            ...
        }
    ]
}
```

New features and enhancements

- **`LauncherWrapper`** is deprecated (**v4.6**)
  - Use the new **`job.launcher.modifier`** and **`job.launcher.modifier_opts`**
- Sanity checking is not required for **`CompileOnlyRegressionTest`** (**v4.6**)
  - The compile stage would fail anyway if compilation fails
  - If a sanity function is provided, it will still be used.
- Scheduler and launcher types can be used as partition constraints (**v4.6**)
  - **`valid_systems = [r'%scheduler=slurm', r'%scheduler=squeue']`**
  - **`valid_systems = [r'%launcher=mpirun']`**

New features and enhancements

- Support multiple inheritance of variables (**v4.6**)
  - Variables defined in a mixin can now be inherited multiple times → increases the composability of tests, which is very useful for libraries
  - The resolution of the final variable value is up to the user
  - The feature is enabled only when **`merge_func`** is passed as an argument to a **`variable`** definition

```
class Mixin(rfm.RegressionMixin):
    x = variable(
        typ.List[int], value=[],
        merge_func=lambda x, y: list(map(max, zip(x,
y)))
    )

class X(Mixin):
    x = [3, 4]

class Y(Mixin):
    x = [10, 1]

class Z(X, Y):
    pass

assert Z.x == [10, 4]
```

New features and enhancements

- More versatile pipeline hook ordering when inheriting tests (**v4.5**)
  - By default, hooks in a stage are ordered in *reverse* MRO order
  - If a hook is marked as **`always_last`** it will be appended at the end of the stage in MRO order
  - This allows hooks of library base classes to always have the last word (e.g., assembling test parameters into command-line arguments of the test's executable)

```python
class X(rfm.RunOnlyRegressionTest):
    @run_before('run', always_last=True)
    def hook_a(self): pass

    @run_before('run')
    def hook_b(self): pass

class Y(X):
    @run_before('run', always_last=True)
    def hook_c(self): pass

    @run_before('run')
    def hook_d(self): pass

# execution order of Y hooks
# X.hook_b, Y.hook_d, Y.hook_c, X.hook_a

# execution order of Y hooks
# in the absence of `always_last=True`
# X.hook_a, X.hook_b, Y.hook_c, Y.hook_d
```

New features and enhancements

- Relative imports in test files (**v4.6**)
  - When ReFrame imports a test file and encounters an **`__init__.py`** file in the directory hierarchy it will load it as a parent module
  - Quite useful for test libraries using utilities where both the library test and the final test need to be able to run

```
~/reframe-examples/howto
├── testlib
│   ├── __init__.py
│   ├── simple.py
│   └── utility
│       └── __init__.py
└── testlib_example.py
```

```python
### test_example.py ###
import reframe as rfm
from testlib.simple import simple_echo_check

### testlib/simple.py ###
import reframe as rfm
import reframe.utility.sanity as sn

# This will fail when loading directly testlib/simple.py,
# unless testlib/__init__.py is present
from .utility import dummy_fixture
```

Future outlook

- Improve reporting and post processing of reports
  - Update **`--performance-report`** output
  - Search and compare easily with past reports
  - Generate references automatically
  - Obtain test references outside test
- Support for incremental test constraints through the **`valid_systems`** and **`valid_prog_environs`**
- Support for custom indexing in references
- Drop support of Python 3.6 after Centos 7 EOL