Experiment Management and Reproducibility

Carson Woods, Derek Schafer, Tony Skjellum

The University of Tennessee at Chattanooga

2021

Background

Many causes of non-reproducible behavior in experiments:

- Unintentional:
  - Different environment (different module files loaded, variances between user software environments, etc.)
  - Variance in data input (deliberate/accidental changes that have gone untracked)
  - And many more…
- Intentional:
  - Running the same experiment on different machines/architectures/software environments.
  - Malevolent changing of inputs/outputs to fit desired results

Spack – reproducible software stacks

- ***The good:***
  - Package specs, hash, variants, etc.
    - Spec specific hash that changes when a package changes (useful on single system)
    - Portable package specs
  - Build-level control:
    - installs each package from source
    - offers near complete control over a package's build time options, version, dependency version, and compiler flags.
  - Spack environments: portably reproducing a software environment that "just works" in the best-case scenario
- ***The bad:***
  - Not entirely reproducible
    - Not every package is available on every system
    - Spack will often take certain liberties when installing software
    - Existing environment variables can bleed into Spack's package installs which can cause inconsistencies that are hard to identify
  - Environments often don't "just work" when using non-default configurations of some packages

Spack – global environments

- Spack was designed with a single user in mind
- Actively working to adopt Spack for use with teams, sites, and projects
  - Users can build against Spack packages that were installed by a system administrator
  - Can allow for quicker iterations of a software environment, without worrying about making the same software available to everyone
  - Aims to be included in the next major Spack release
- More details in:
  - Woods, Carson, Curry, Matthew L., & Skjellum, Anthony. (2019). Implementing a Common HPC Environment in a Multi-User Spack Instance. Presented at the SC19 (HPCSYSPROS19), Denver, CO: Zenodo. http://doi.org/10.5281/zenodo.3525373

Runtime Environment Capture (REC)

- Current operates as a python script wrapper around an existing command:
  - python rec.py [rec_arguments] [script]
- Current state of REC:
  - Captures start time / end time of job
  - Supports various methods of job launching (cli, shell, slurm, sge, etc.)
  - Captures launcher and launcher version
  - Captures executables included in scripts and their self-reported versions
  - Captures SHA256 hash of input script/command
  - Captures stdout

REC – Future Work

- Short Term Goals
  - Application refactor
  - Improved failure detection
  - Spack environment integration
  - User environment capture
  - Library capture: capturing name/version of libraries that executables link against
- Long Term Goals
  - Profiling tools (perf, strace, OpenXDMoD, etc.)
  - Ptrace
  - Input/output tracking
  - Better job launcher integration
  - Better diff utility for reproducibility reports

Conclusions

- There are many existing tools that can assist with reproducibility
  - Not built for use in ensuring reproducibility
  - Complex and implementing all of them can add significant overhead
- REC attempts to facilitate this by bringing together these existing tools together in a customizable and lightweight wrapper around existing experiment workflows
  - Minimal overhead with little to no “invasive” changes needed
  - No containers required

Questions?