A Hybrid Approach to Scientific Software Package Management on an HPC Cluster

Qiyang Hu & Shao-Ching Huang

Institute for Digital Research & Education, UCLA

November 17th, 2020 • Atalanta, GA

Outline

1. Challenges for software managements for HPC clusters
   - Traditional “configure-make” manual ways
   - Using package management framework
2. Our practice at UCLA
   - Defining customized module file structures
   - Hybridizing module files
   - Operational workflow
3. Discussions and summary

Challenge on managing software packages manually (in-house)

- Combinatorial Complexity in HPC environments
  - UCLA: > 10,000 combinations

- Tedious and time-consuming processes
  - Keeping the deployment up to date
    - new versions of compilers, libraries and the software itself are released
  - Supporting users for various usage problems
    - platform/compiler/library-dependent compilation issues
    - linking/runtime issues
    - sub-optimal application performance issues
  - Instructing users to do the installation in their user space
- Hard to keep consistent over multiple system admins
  - Naming schemes (software, module names, etc)
  - Tree structure of software sets, tools and modules
  - Up-to-date documentation

A high-level framework as a solution

- Framework ≈ a package manager
  - "High-level" means to manage software applications in a systematic manner
    - Automatic build tools
      - Use existing Make, Autotools, CMake
      - Handle dependencies, various flags
    - Organize/present for easy use to end users
      - Environment Module System, Lmod
  - Good to serve for
    - system admins
    - HPC user support team
    - general users: developers, scientists
- Spack and EasyBuild are two common choices
  - UCLA: we select Spack
    - Flexibility
    - Successfully installed ~300 pkgs using Spack

Challenges for adopting a software management framework in practice (1)

- Observed lots of “seemingly” duplicated copies of installed packages
  - Natural consequence of systematic management for combinatorial situations
  - A simple package name cannot contain all info about dependencies and flags
  - Spack uses appending hashcode in every package name
  - Confusions/inconvenience for sysadmins and users

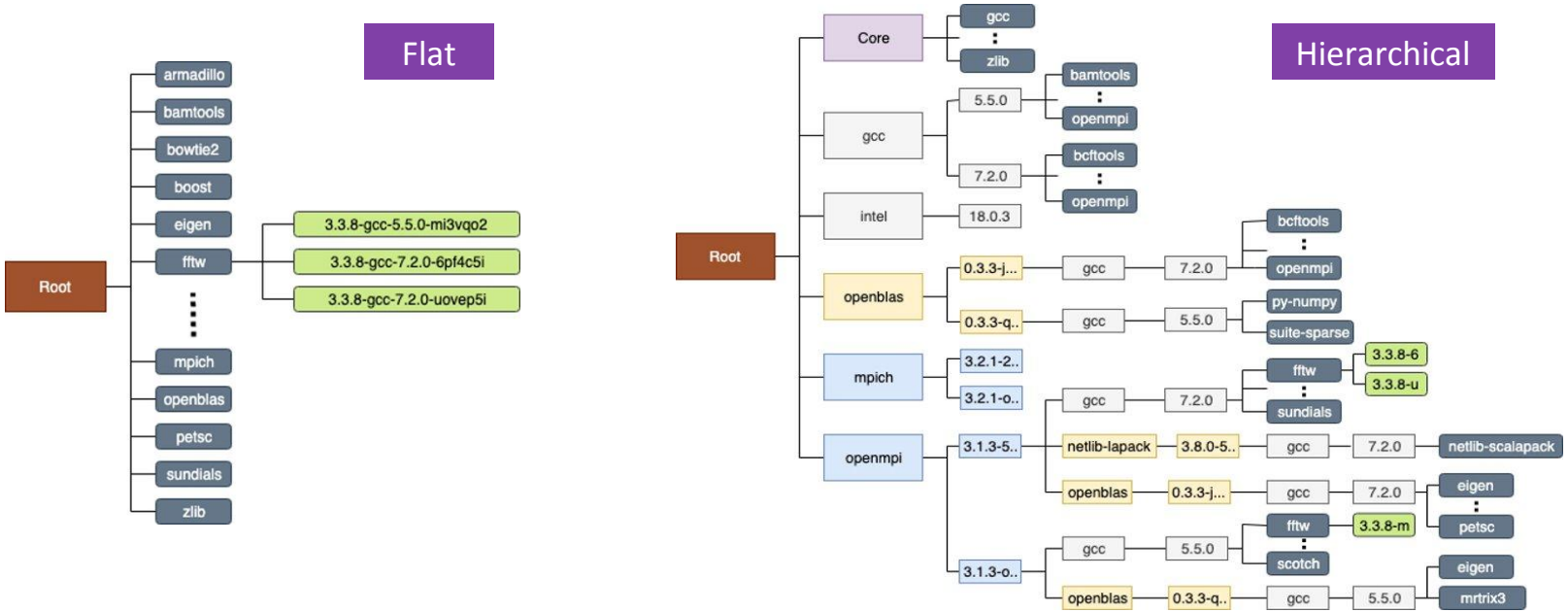
```
$ module spider hdf5

----------------------------
  hdf5:
----------------------------
     Versions:
        hdf5/1.10.6-kvou6zc
        hdf5/1.10.6-vgwabjx
        hdf5/1.10.6-vsnvqrh
        hdf5/1.10.6-3gjqlqm
        hdf5/1.10.6-4icro2m
        hdf5/1.10.6-zjhc7ms
```

4 hdf5 copies under gcc/8.3.0 and openmpi/3.1.5

```
hdf5@1.10.6%gcc@8.3.0~cxx~debug+fortran+hl+mpi+pic+shared~szip~threadsafe
hdf5@1.10.6%gcc@8.3.0~cxx~debug~fortran+hl+mpi+pic+shared~szip~threadsafe
hdf5@1.10.6%gcc@8.3.0~cxx~debug~fortran~hl+mpi+pic+shared~szip~threadsafe
hdf5@1.10.6%gcc@8.3.0~cxx~debug~fortran+hl+mpi+pic+shared+szip~threadsafe
```

Challenges for adopting a software management framework in practice (2)

- Found it non-trivial to accommodate the existing (legacy) scheme
  - Spack provides modules.yaml and configure.yaml
  - But not flexible enough, e.g. re-defining hierarchical modulefile structures



Challenges for adopting a software management framework in practice (3)

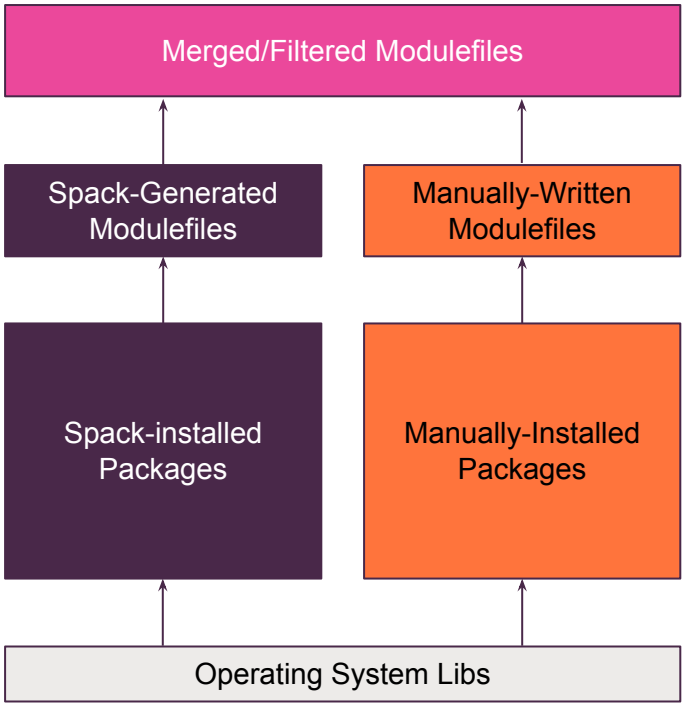
- People (sysadmins & users) appear to be reluctant to adopt.
  - Overwhelmed by the complexity controlled in a “Spack-ish” way
  - Huge momentum to keep the existing scheme/work
  - Are cautiously waiting for further updates
  - OK to tolerate sub-optimal runtime performance in university environments

Our Goal: to make the Spack and existing manually-installed systems ***co-exist***

- For sysadmins: workflow scriptable as much as possible
- For end users: package list presented as clean as possible

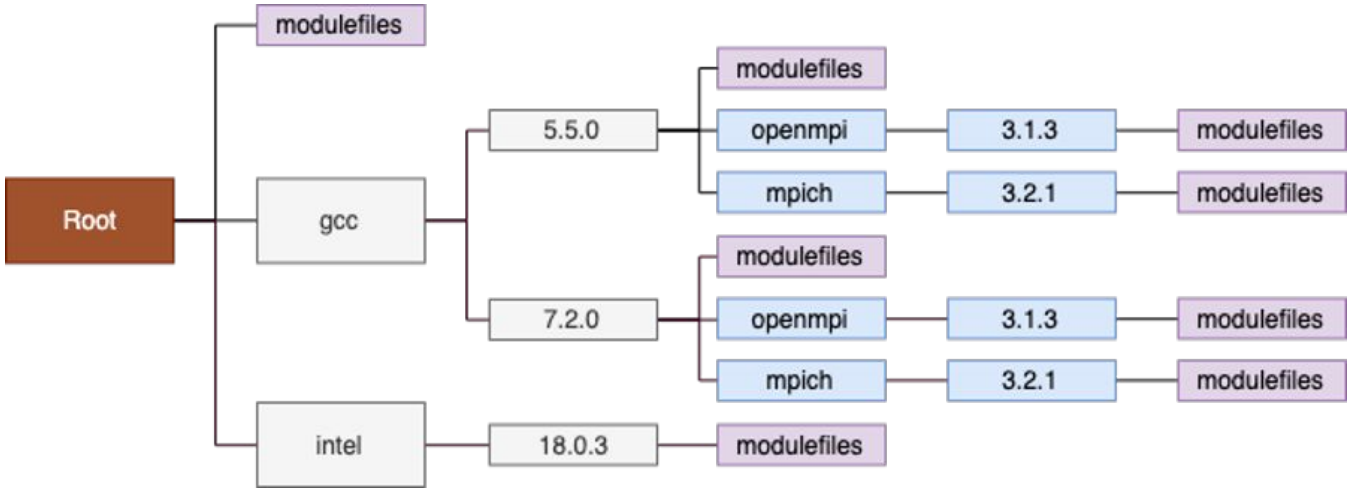
Our Central Idea: introduce a top layer above both systems

- Keep Spack and Non-spack systems de-coupled as possible
  - Software libs, binaries
  - MODULEPATH(s)
- Main work on merging modulefiles
  - Take advantage of Spack-generated module files
  - Re-shuffle/sort module files from both systems
  - Introduce more controls to customize the presentations for user-support needs



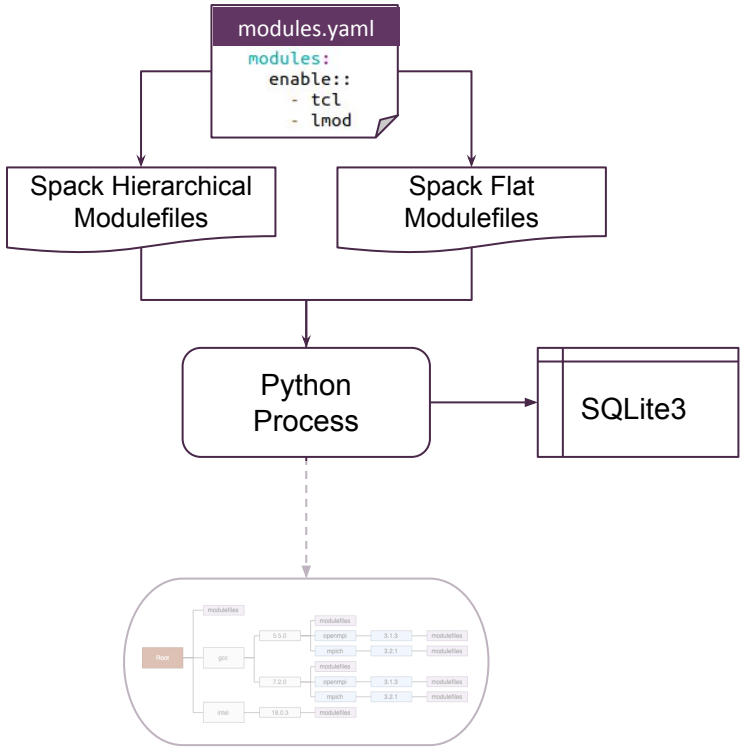
Our practice at UCLA: a prototype of the customized modulefile structure

- Root = Default End-user MODULEPATH
- Hierarchical



Our practice at UCLA: hybridizing modulefiles (1)

- Spack generates both hierarchical and flat structured modules
  - Hierarchical: Lmod
  - Flat: TCL
- Parse out package info from Spack-generated hierarchical modulefiles
  - Name, version
  - Compilers
  - MPIs
  - Other flags
  - Hash code
  - folder/absolute paths
- Save info to an SQLite3 file (`apps.db`)
  - Fields: corresponds to the customized modulefiles
  - Save path info of flat-structured modules

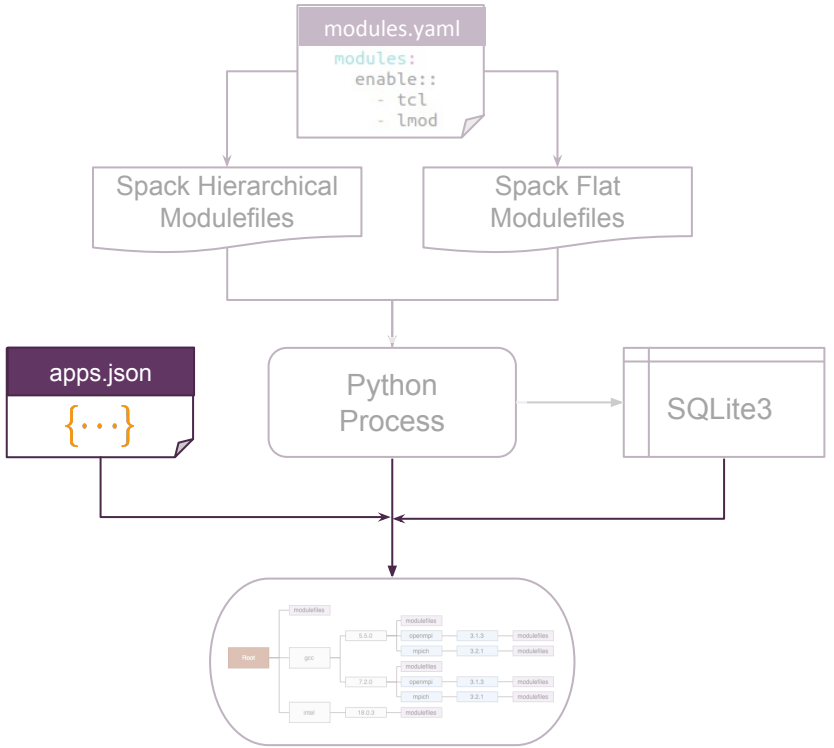


Our practice at UCLA: hybridizing modulefiles (2)

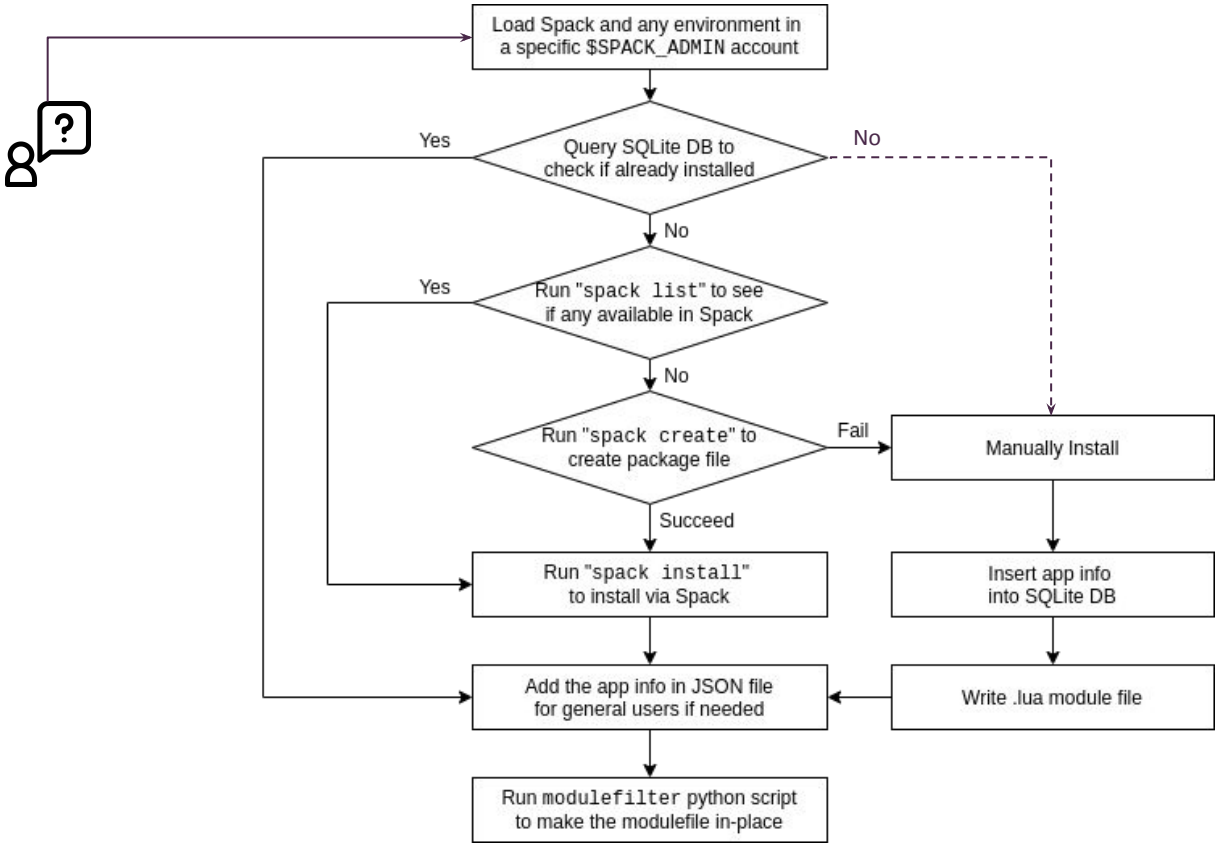
- `apps.json` to define the support range
  - Input from sysadmins

```
{    "bowtie2": {
         "name_in_spack": "bowtie2",
         "version": [
             "2.3.4.1-gcc-7.2.0",
             "2.3.4.1-gcc-5.5.0"
         ],
         "default": "2.3.4.1"
     },
}
```

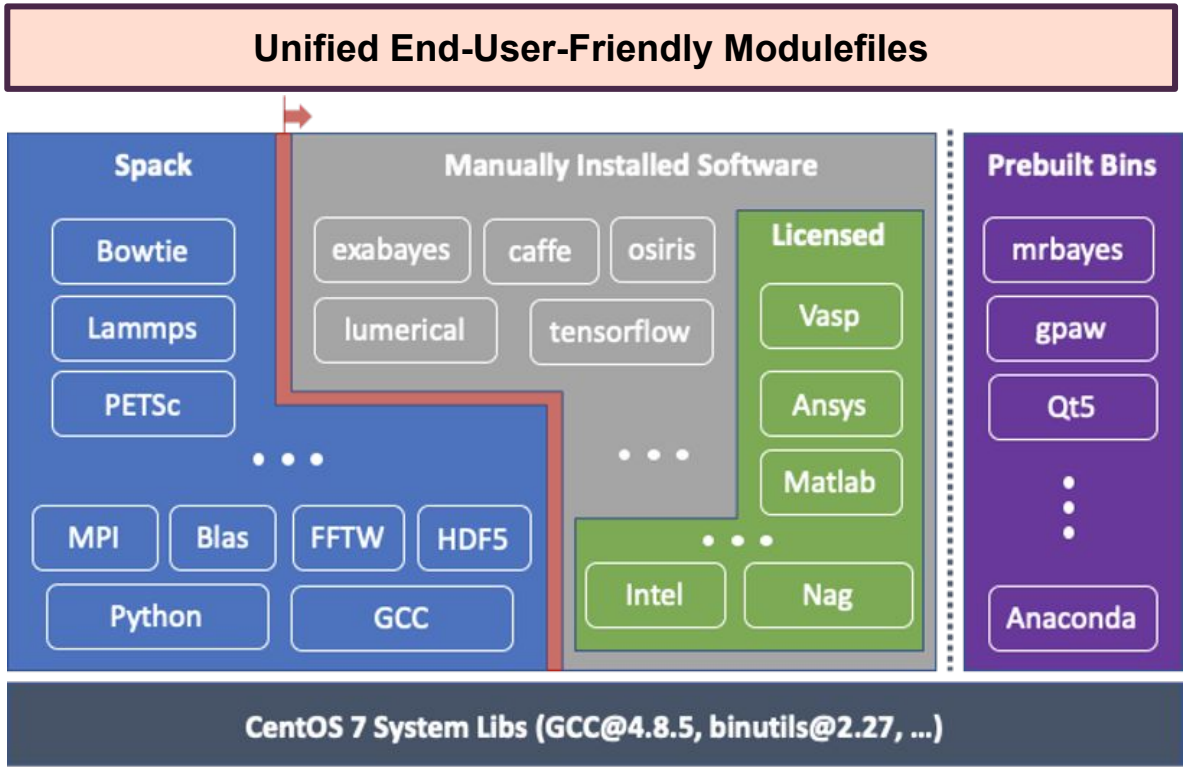
- Processes include:
  - Convert TCL/C to Lmod
  - Pre-pending MODULEPATH for maintaining customized hierarchies
  - Adding “family”
  - Specify default version
- Copy flat structured modulefile to destinations.



Our practice at UCLA: Operational workflow



Our practice at UCLA: a proposal



Discussion and Summary

- Trend to keep framework-based and customized software managements ***co-existing***
  - Many pre-existing (legacy) HPC systems
  - Spack provides more support for linking to external pkgs
- Our approach
  - Introduce a top layer to unify software systems
  - Help sysadmins adopt the framework solution
  - General enough to be not limited to Spack
- Possible future extension
  - Spack, Manually-installed, Prebuilds, Containers
  - Manage environments defined in Spack & Conda
  - JSON file used for website documentation pages

Check Out Our Demo Code

https://gitlab.idre.ucla.edu/hoffman2/modulefile-processing