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

Qiyang Hu
Institute for Digital Research and Education
University Of California, Los Angeles
Los Angeles, California, USA
huqy@idre.ucla.edu

Shao-Ching Huang
Institute for Digital Research and Education
University Of California, Los Angeles
Los Angeles, California, USA
schuang@idre.ucla.edu

***Abstract*—We present a practical approach for managing the scientific packages in the HPC cluster environment of a research university environment that has a diverse user base. The primary goal is to minimize the HPC operational team's burden of installing and maintaining a large number of software packages to support the broadband spectrum of the university's computational research. We propose a hybridizing management method that can harness the power of modern software management frameworks and the flexibility of in-house, or "manual", installation sources, and at the same time present a coherent view to the end users. The Spack [1] framework was used in this work. Our hybrid approach is applicable to using other framework tools. The manipulation of Spack-generated environment module files and the typical workflow are illustrated and discussed based on our use case.**

***Index Terms*—software management, environment modules, Spack, HPC**

I. Introduction

A recent challenge for HPC system administrators to adopt the software installation frameworks (e.g. Spack [1] or EasyBuild [2]) is that those frameworks may have to be integrated with in-house package installation practices already existed. In practice, the in-house installation component is required because not all packages can be satisfactorily managed within a software management framework.

In this work, we propose a practical approach for HPC cluster operational teams to manage software packages by hybridizing the framework and manual-based installation methods. Conceptually, our central idea is to add a layer above the existing software management frameworks, where a workflow is introduced to manage applications built either by the framework or by manual installation to enable flexible customization, complying to the existing structure of the HPC cluster environment. The framework-installed and manually-installed packages are kept separate by design, so it will allow future changes of the HPC environment, including the possibility of adopting another installation framework. In this paper, we adopt Spack as the package management framework. The proposed approach is applicable to using other software management frameworks. We consider the deployment of the hybrid approach on UCLA Hoffman2 Cluster where an in-house package installation structure already exists. A main contribution of this work is a novel concept to manipulate and re-organize the environment modules and offer a practical, automated and script-able workflow to build and maintain the packages, and to provide a coherent view to end users.

II. Modulefile structure cutomization

Adding a flexible management layer on top of framework requires traversing the module file structure generated by the software management framework. From our experiences in practice, configuration through Spack's YAML files is not flexible enough for our needs. Specifically, Spack's installation processes create multiple duplications of installed packages and libraries. Presenting all versions of the Spack-built software packages, with embeded unique hash codes in the package's display names at once, introduces big confusion to the end users. In addition, the predefined hierarchical structure create by Spack is quite rigid and incompatible for our existing cluster environment. There seems to be no easy way to customize it.

Based on our experience, also motivated by the operation of other large-scale HPC clusters (e.g. TACC Stampede), we first propose the module files to be organized in a simple structure as shown in Fig. 1, where the MPI level is nested in the compiler level. The versions of compilers and MPIs follow the convention in Spack. Additional categories can be nested in Compiler level or Compiler+MPI levels if needed. Such a hierarchical organization, in contrast to a flat one, can prevent the users from loading incompatible software packages (e.g. libraries compiled with different MPI implementations) in practice.

III. Hybridization of modulefiles

With our own design of module file structure defined in Section II, hybridization of the framework-based (Spack) and the manually-installed systems is achieved by implementing a configurable procedure to (1) traverse the modulefile tree generated by Spack, (2) filter out the undesirable packages (e.g. duplicates and not-to-be-supported packages), and (3) export the results to a new modulefile tree which will be the default module path for end users.

We use JSON to express the information of the packages that our HPC operational team will support, i.e. visible to end users. An example of a JSON file is listed below:

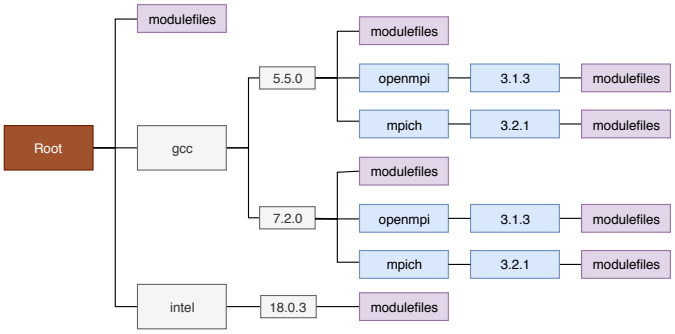

Fig. 1. Newly defined hierarchical structure of module files in Hoffman2

```
{    "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"
     },
}
```

Our approach takes advantage of Spack's capability of generating both flat and hierarchical module files by enabling them in modules.yaml configuration file. As shown in Fig. 2, we first traverse over Spack's hierarchical `Lmod` [3] module structures to obtain the hierarchical information about the installed packages and the hash codes generated from Spack. All collected information is stored in a SQLite database. Then we traverse Spack's flat-structured module files to find the ones in the database with the same hash code. By cross-checking the customized JSON file to decide the display option for the packages we can create the final modulefile tree in the destination directory. This process specifically include the following steps:

- Using `tcl2lua.tcl` script to convert the `TCL/C` module file to `lua` format.
- Pre-pending `MODULEPATH` to enable the dynamically showing of available module files under the corresponding Compiler/MPI directories if necessary.
- Adding `family` to guarantee users can only load one compiler or MPI stack at a time, if necessary.
- Specifying the default version for the packages by generating `.version` file in the destination package modulefile folder.

In the SQLite database, (`apps.db`), there is currently only one table, (`apps`). The schema of the table can be reviewed by `.schema apps` inside the SQLite shell. The detailed scripts is publicly available in a gitlab repository [4].

IV. Operational Workflows

The workflow to hybridize the Spack-installed and manually-installed packages as described in Section III is illustrated in Figure 3, showing an example of the process of installing a package on the HPC cluster for a system admin, starting from recieving an installation request, logging

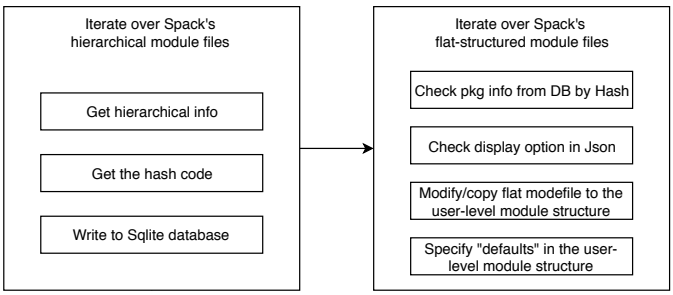

Fig. 2. Flowchart for the python script to generate the newly defined module file structures for user-level presentations

into a dedicated account (`$SPACK_ADMIN`) and ending with displaying the module(s) to end users. It should be noted that the workflow can be also taken in any subgroup package environment defined in Spack [5].

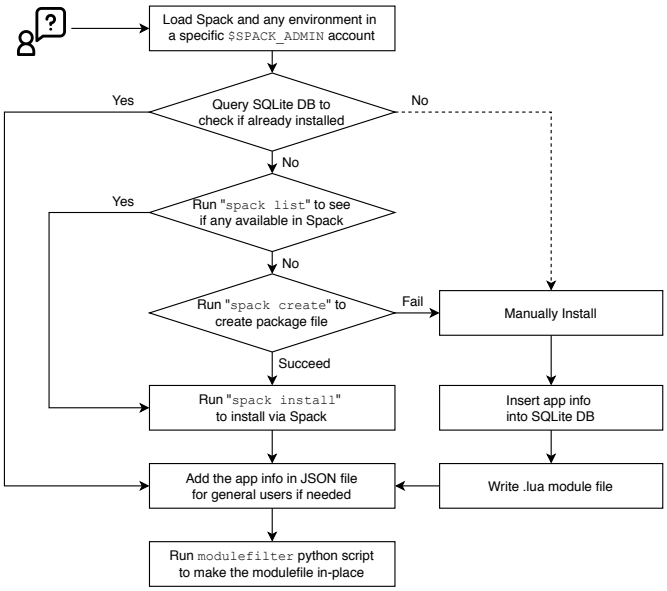

Fig. 3. Workflow to hybridize the Spack-installed and manually-installed packages for an administrator to install a requested package

V. Summary

We propose an approach to hybridize the framework-based and the manual-based software management systems, and present a proposed workflow to be able to process, filter and integrate the two systems to any target modulefile presentation.

References

[1] T. Gamblin, et al. "The Spack package manager: bringing order to HPC software chaos" In SC '15 Proceedings of the International Conference for HPC, Networking, Storage and Analysis, 2015.
[2] M. Geimer, et al. "Modern Scientific Software Management Using Easy-Build and Lmod" In HUST '14: Proceedings of the First International Workshop on HPC User Support Tools, page 41-51, November 2014.
[3] R. McLay, et al. 'Best practices for the deployment and management of production hpc clusters" In SC '11 In State of the Practice Reports, page 9:1–9:11, 2011.
[4] https://gitlab.idre.ucla.edu/hoffman2/modulefile-processing
[5] https://spack.readthedocs.io/en/latest/environments.html