

# Optimizing experiments' software stack management with Spack

Paul Chelarescu

Supervisors:

Javier Cervantes Villanueva

Graeme Stewart

Patricia Mendez Lorenzo

## Abstract

Spack is a package manager for supercomputers with a high degree of flexibility and specificity for managing software installations and with a focus for scientific applications. Spack is currently being used at CERN for building the Future Circular Collider (FCC) software stack. This project presents the improvements made to Spack for speeding up the build time and the upgrades which proceeded for bringing the compiler version used up to date.

## Introduction

A package manager is a collection of software tools used for installing, removing, upgrading and configuring software packages in a consistent manner. Packages are collections of source files, metadata and lists of dependencies necessary for complete functionality. In addition, the main functionalities of a package manager are extracting the archive files, ensuring integrity across an installation tree and avoiding dependency issues created by incompatible versions of libraries. The context of High Performance Computing (HPC) software introduces further challenges on top of the ones mentioned above. Firstly, software installations on HPC machines are massively combinatorial in nature, having thousands of combinations that appear by choosing between different package implementations, different versions amongst those packages, number of types of platforms and compilers and number of compiler versions, among others. The challenge is having a consistent environment across different machines with reproducibility as a main concern, similar in nature to the High Throughput Computing service at CERN ([similarities](#)). In these conditions, manual deployment is not scalable and automated tools become a necessity.

## Structure of Spack

Packages are represented inside Spack as Python classes which contain the instructions for installing the package given the destination environment. This includes dependencies with version ranges, patches, CMake arguments, enabling and disabling options for the package and describing variants of the package. The Spack database is in JSON format and specifies the dependencies of every package installed, its location and any additional attributes such as compiler flags. Every package is located inside a prefix containing the hash of the package in the path, therefore enabling any number of package combinations to co-exist on the same machine at the same time. Package dependencies are represented as a Directed Acyclic Graph

and spack ensures consistency inside this installation tree. For example, if there exists package A and package B which both depend indirectly of package C, Spack will ensure that package C will only be installed once and with a version that is compatible with both A and B.

## **Improvements to the current infrastructure**

Following are a list of improvements that have been made over this summer to the Spack infrastructure at CERN:

### **Addition of a package recipe for Spack [\(1\)](#)**

Spack now contains a recipe for the package Minuit, a numerical minimization program.

### **Ccache [\(2\)](#)**

A compiler cache for recompilation, ccache works by caching installations and detecting when they are being done again. Ccache is currently integrated with Spack using simple configuration with yaml files. A simple benchmark can show that in practice, enabling ccache can provide up to 30% speedups in compilation time for the FCC software stack, from 52 minutes to 37 on a machine with 24 CPUs Intel(R) Xeon(R) E5-2630L @ 2.00GHz with 64 GB of RAM. Furthermore, the integration with Jenkins uses a Master-Slave architecture with a last used node distribution algorithm which tends to localise the cache on the same machines, further adding to the benefit of caching builds.

### **Chaining Spack installations [\(3\)](#)**

Spack can now chain installations across different file systems by linking against an upstream package database, thereby removing the need for recompilation of existing packages. In practice this can lead to a reduction in time of over 100 times since the packages will only be scanned and linked as opposed to compiled every time.

### **Upgrading the GCC compiler [\(4\)](#)**

The GCC compiler used for the FCC stack was updated from version 6.2 to version 7.3 as part of this project. The main changes involved have been making new configuration files, patching build files, configuring C++ standard flags in Spack recipes, submitting bug fixes to the Spack repository as well as investigating archive, variants and dependency incompatibilities.

### **Migrating the setup scripts from bash [\(5\)](#)**

Previously, the main functionalities providing the preparation of the environment for FCC have been done through bash scripts. Most of the functionality has been moved into Python modules, with the added benefit of code clarity and maintainability.

## **Conclusions**

Spack has benefitted from many improvements over the course of this project and the integration with the build infrastructure, thereby providing a better environment for building and managing the software used in the CERN experiments.