Paper TT05

HPC / SLURM + Open Source Clinical Trial Workflows: Current State

Michael Mayer, Posit PBC, Allschwil, Switzerland
Phil Bowsher, Posit PBC, Indianapolis, USA

ABSTRACT

SLURM (short for Simple Linux and Unix Resource Management) is a popular resource manager on HPC platforms. HPC clusters are heavily used in clinical trial workflows given their great scalability. With the ever increasing complexity of scientific software and the need for software validation, containers are becoming increasingly important in HPC. In this paper we will summarize the current state of the use of SLURM + Containers in clinical workflows

INTRODUCTION

SLURM is a popular Resource manager on HPC clusters that are used for clinical trial workflows. Many organizations distribute workflows into a queue and manage orchestration for containers with Singularity/Apptainer. Posit/RStudio will be presenting an update regarding SLURM/HPC in clinical trials environments with Posit Team. This talk will discuss opportunities and applications for SLURM to empower stakeholders, open source administrators, and statistical programmers. This paper will explore and discuss areas such as HPC and containers and where they are being used, terminology, current research, etc.

WHY ARE HPC CLUSTERS SO PREVALENT/USEFUL IN CLINICAL WORKFLOWS ?

In general most parts of a clinical workflow do not need excessive compute power. This includes general data wrangling/cleaning processes, basic/standard analyses that produce the usual TFLs (Tables, Figures and Listings) that comprise the document sent to the health authorities for a new drug application.

There is a few parts however where significant compute power can be very helpful:

- **Clinical trial simulations:** Once a promising new drug/pathway has identified that could help cure a disease, a clinical trial needs to be designed. The outcome of this trial needs to show that the designated endpoints are met. Given the fact that running clinical trials are very costly for a pharmaceutical company, the goal is to enroll the optimal number subjects to produce a statistically significant result against the defined end points.
- **Parameter studies:** Sometimes models can become very complex and while a fit against the data produces a result, it is not clear how meaningful/reliable this result is. In order to assess this, technologies like bootstrapping, and general parameter studies are being used.
- **PK/PD models:** While standard one to three compartment models are easy to compute, more sophisticated models can increase computational demand significantly.

Two main programming paradigms can be observed:

- **"embarrassingly parallel"** or Map Reduce type of workloads (e.g. parameter studies – run the same simulation 10^3 times with slightly perturbed parameters/initial conditions) – these are many computations that can happen in parallel but do not depend on each other.
- **Massively parallel** workloads: A single computation is taking so long that distributing the tightly coupled computations over many cores will help reduce elapsed time and speed up the computation. In many cases this brings the model into the realm of feasibility while without it such models would take way too long. Examples for this type of workload include certain PK/PD models, large/complex Stan calculations, …

The above points strongly to the use of an HPC where compute power can easily be leveraged in order to speed up computations so that the results are obtained in a reasonable amount of time. Some of the use cases mentioned above also need iterative computations where the result of one run needs to be analyzed and will inform the next computation.

WHAT ABOUT CONTAINERS ?

While the use of docker containers is a de facto standard in Kubernetes based environments, on an HPC the use of docker containers is rather problematic due to the need for elevated permissions.

There is a few different container technologies available that are suitable for HPC such as shifter, singularity and apptainer (the latter a fork of singularity). Those container technologies all work in userspace.

Containers today are used for many reasons, one of them is the need for reproducibility. In the context of clinical work, a container will contain all versions of the desired open source software (e.g. R and python) and also all the needed and tested/validated packages baked in. Given that such a container is immutable by the user, once the container and its software has been tested/validated once, the consumers can use any of the installed software and packages without any further testing and write their code.

While containers are undeniably used for increased reproducibility, the main benefit is to have an immutable and fully tested set of software and versions available. Like with software validation typically using a risk-based approach, it has to be pointed out that strict reproducibility is only guaranteed if the containers are run on the same hardware. If the same container is run on different processor generations, the binary code can produce numerically inconsistent results if not managed properly (cf. Intel MKL and Function Multiversioning).

What about Open Source Workflow Tools ?

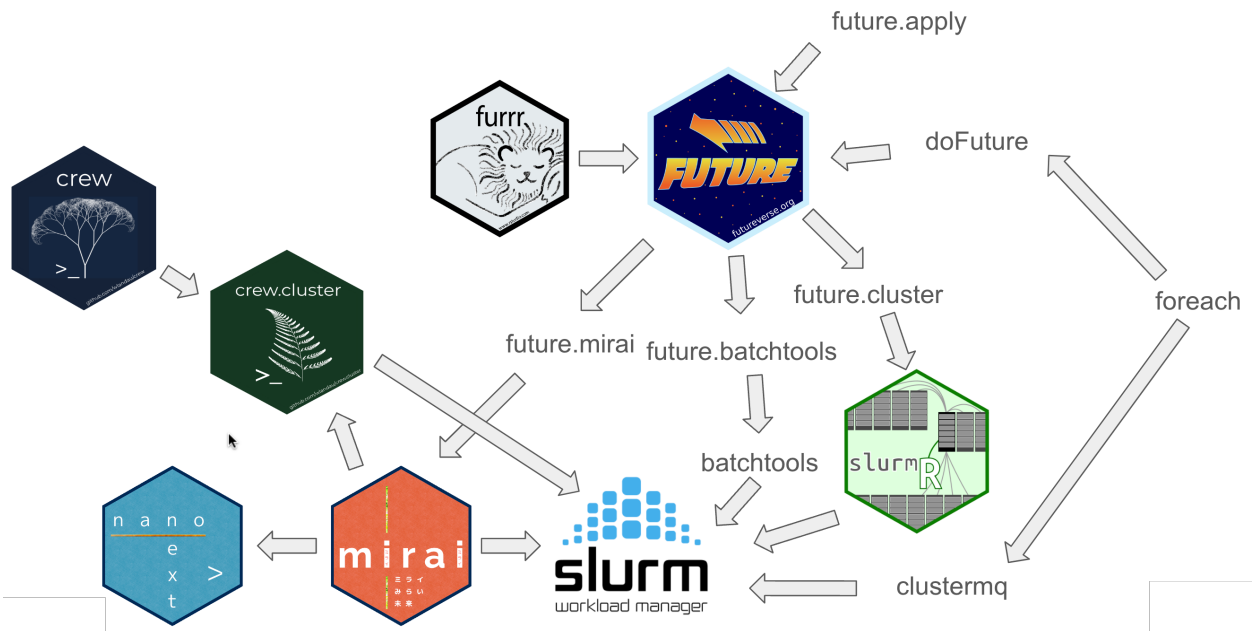
There is a few workflow tools out there:

- Apache Airflow
- Nextflow
- luigi
- SnakeMake
- GNU make
- targets

Some companies seem to have good results using Snakemake, other companies are exploring the use of Apache Airflow and NextFlow. With the exception of targets that uses R, all of those tools are language agnostic.

Usage of R on SLURM based HPC clusters

There is a whole eco system of R packages available for High Performance Computing in general. Not all of them work on SLURM based HPC cluster. Here we would like to highlight a few examples of modern R packages that are known to work with SLURM and are frequently used. The below diagram shows them and how they relate to SLURM. Where available, the hex sticker was used.



For example and to help with reading the diagram, the foreach package (used for parallel for loops) can be used both with doFuture and clustermq. Clustermq itself interacts directly with SLURM via its templates while doFuture uses the future package. Future package then interacts with SLURM via one of its many backends and so on.
The list is definitely not complete, it however depicts a number of R packages that are fairly modern and also are very efficient when it comes to parallel computing. This is achieved by the use of NNG with nanonext being the R interface for it and NNG's predecessor, the ZeroMQ framework that are particularly efficient for network communication. Batchtools and the cluster backend to future are a bit outliers here. In contrast to all the other packages mentioned that are mostly running in-memory, batchtools heavily relies on disk storage for some transient data which creates quite a bit of overhead and can lead to reduced performance and scalability. The cluster backend of the future package relies on the cluster objects of the parallel package. While they definitely have their use cases where they

perform well, given the fact that they are missing more modern communication frameworks, their scalability is typically very bad.

USAGE OF CONTAINERS ON HPC

While there is shifter, apptainer and singularity out there, it seems that the most popular choice is apptainer and/or signularity (the latter also has a commercially supported professional version while apptainer is purely open source). For the following discussion we exclusively focus on apptainer and singularity. Given the fact that the functionality is still very similar despite the earlier fork, we use apptainer and singularity interchangeably.

Containers can be run in multiple ways on a SLURM based HPC. The singularity/apptainer CLI can be called directly in a SLURM sbatch script or deeper integration via the so-called SPANK architecture of SLURM is possible, e.g. via a singularity plugin.

If a pharma company is validating their set of required packages, a rather complex process of risk-based validation is triggered. More information can be found on the R validation hub. For the discussion in this paper, we simply assume that there is a process that will eventually give validation status to the desired packages.

At some point those packages in their respective versions have to be baked into a container. Two approaches are possible:

- Build a docker container and then convert it into a singularity image: Building docker containers makes the development experience much better since the docker layers are cached and hence the iterative debugging is much faster. The disadvantage is that you then still need to convert the docker container to a singularity image.
- Write a singularity recipe: This ensures that you directly get a singularity container as an output. You also do not have issues with elevated permissions etc. as everything can be run user-space.

As part of your container build, you will not only need to install the software (e.g. R or Python) but also all the packages that you desire and you are validating. This installation can be a very time-consuming process as `install.packages()` in R and `pip install` in Python are running sequentially. On the R side you also have to deal with additional operating system dependencies for your R packages. Fortunately on the Python side this is not that much a problem as long as there is python wheels (or conda binaries) available.

For R package installations, we highly recommend the use of pak as an alternative to `install.packages()`. Pak does not only run the download, build and install of packages fully in parallel, it also auto-detects missing OS dependencies and installs them when run as root (which is typically the case when building containers). Like renv it also creates a lock file that contains all the metadata needed for full reproducibility.

Similarly on the python side there is uv that also significantly speeds up python package installations.

An example for r-session-complete type of containers built for the use in Posit Workbench via the SLURM Launcher shows the use of pak. The main R code to be run is

```
# creating a lockfile to record all needed metadata for reproducibility
pak::lockfile_create(packages_to_be_qualified,
       lockfile=paste0(site_library,"/pkg.lock"))

# use the create lockfile to install packages
pak::lockfile_install(lockfile=paste0(site_library,"/pkg.lock"),
       lib=site_library)

#alternatively you can directly install packages
pak::pkg_install(packages_to_be_qualified, lib = site_library)
```

`packages_to_be_qualified` is a lis of R packages including their versions in a pak compatible syntax.

The use of pak for R as well as uv for Python can also be very helpful when running CI/CD pipelines, reducing the time needed to build environments on-the-fly considerably.

When working with a customer that wanted to install 1500 R packages each for 2 R versions into their singularity container, they ended up with a build time of more than a day when using `install.packages()` - with the use of pak we were able to reduce the build time to less than an hour.

ACKNOWLEDGMENTS

The authors explicitly acknowledge the high quality R software developed that continuously enriches the R ecosystem and facilitates the use of High Performance Computing - in particular, Will Landau for building and maintaining crew, crew.cluster and targets, Charlie Gao for his work on nanonext and mirai, Henrik Bengtsson for the futureverse and the recent addition of the future.mirai package, Michael Schubert for clustermq and Michel Lang for batchtools.

Michael Mayer also would like to thank Phil Bowsher for submitting the abstract. Without this, this paper and the presentation would have never happened.

CONTACT INFORMATION

Author Name: Michael Mayer
Company: Posit PBC
Email:michael.mayer@posit.co
Website: www.posit.co

Author Name: Phil Bowsher
Company: Posit PBC
Email: phil@posit.co
Website: www.posit.co