Analysis Facilities for Late-Stage Analysis

Kevin Lannon (klannon@nd.edu)[1], Paul Brenner[1], and Mike Hildreth[1]

[1]Univeristy of Notre Dame

1 Introduction

A consistent trend in particle physics data analysis has been increasing data velocity, that is the rate at which data are accumulated over time. To keep pace with data, physics analysts must be able to turn around analyses on increasingly large datasets. Traditionally, HEP has been fairly adept at deploying sophisticated distributed computing techniques, relying on grid and batch computing models to keep pace with growing data velocity and volume. However, in the foreseeable future (for example, HL-LHC), the challenge will grow to the point at which even analysis techniques such as late-stage production, visualization, and statistical interpretation of histograms from a compact analysis "ntuple" format will no longer be feasible with traditional "interactive" resources (e.g. laptops or shared, interactive analysis servers). Instead, users will require resources that enable them to deploy the power of distributed computational resources without the complexity and logistical overhead typically associated with batch processing. We use the term "analysis facility" to describe resources configured in this way. Although the term "analysis facility" evokes a concrete computational resource, the reader is encouraged to view it equally as a set of services and infrastructure that could be concretely realized in a variety of different ways.

2 User Analysis Environment

A growing trend in particle physics analysis software is to take advantage of the robust and mature scientific python ecosystem. Basis especially late stage analysis tools (histogramming, statistical analysis, etc.) on scientific python allows HEP analysts to benefit from the best of both the broader scientific and data science community tools (`numpy` [1], `pandas` [2], `matplotlib` [3], etc.) as well as traditional HEP software, such as `ROOT` [4] through, for example, the `PyROOT` interface. The recent popularity of python-based HEP analysis tools such as `uproot` [5], `awkward` [6], and `Coffea` [7] attest to the power of such an approach. Furthermore, focusing on python-based analysis approaches provides easy integration between HEP analysis and the many popular machine learning frameworks available in python (e.g. `scikit-learn` [8], `tensorflow` [9], `pytorch` [10]). Finally, notebook-based approaches, such as Jupyter Lab [11], are becoming increasingly popular interfaces for this stage of data analysis.

The approach described above enables physicist to take advantage of a broad array of analysis tools shared with the scientific and data science communities. This is a great strength but also offers a significant challenge. The python scientific and data science ecosystem provides the greatest value when python modules can be combined in many different combinations. Furthermore, many of the underlying libraries and tools are under active development with frequent new versions released. To

maximize its potential, the analysis facility should support analysts to deploy a variety of python modules and versions.

For the sake of further discussion, we define a user analysis to consist of a user analysis library and one or more python notebooks, supported by a collection of python dependencies. Tools such as `conda` [12] and `pip` [13] make it possible for users to specify their desired environment in a reasonably compact and portable way. Leveraging technologies like containerization as well as tools like `repo2docker` [14] (one of the underlying technologies for the Binder service [15]) would allow the analysis facility to serve the needs of a many users with a diverse set range of analysis software.

3 Infrastructure and Services

To implement an analysis facility like that described above, there are two main types of infrastructure required, one that is persistent on the timescale of a user session and one that is transient, to provide burst capacity when large-scale parallelism is required to complete an analysis step (e.g. processing ntuples into histograms). An example realization of this infrastructure is shown in Fig 1.

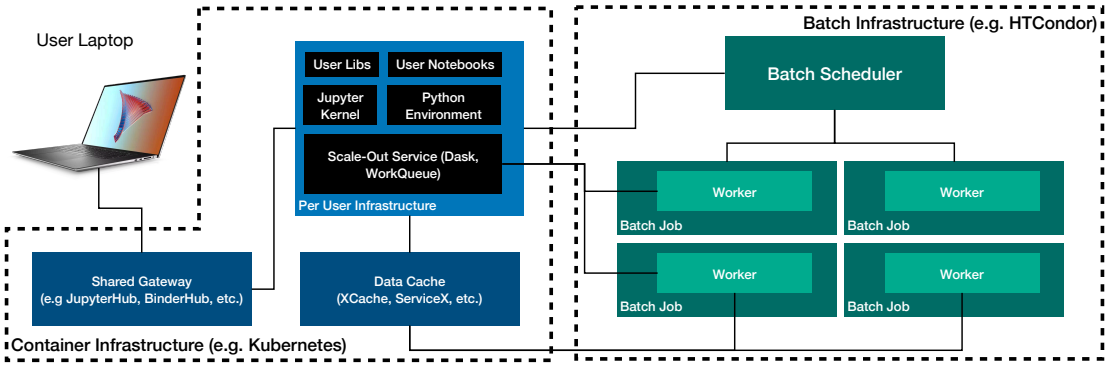

Figure 1: Infrastructure involved in an analysis facility.

The persistent components are materialized in the "Container Infrastructure" on the left-hand side of the diagram. This infrastructure allows the deployment of long-term services like a shared gateway for providing authentication and handling connections to the notebook kernels and caching infrastructure providing fast access to data. This infrastructure also allows the provisioning of resources to support individual user sessions, including the Jupyter kernel, local processing power, and services to support scale-out via such tools as Work Queue [16], Dask [17], Parsl [18], or FuncX [19]. These scale-out services connect to worker processes instantiated in the transient resources, represented by the batch infrastructure on the right side of the diagram. The batch infrastructure provides computational resources to parallelize analysis tasks. The workers in the batch system access data by connecting to the caching services in the container infrastructure.

References

[1] Numpy. `https://numpy.org/`.

[2] Pandas. `https://pandas.pydata.org/`.

[3] Matplotlib. `https://matplotlib.org/`.

[4] Root. https://root.cern/.

[5] Uproot. https://github.com/scikit-hep/uproot.

[6] Awkward. https://github.com/scikit-hep/awkward-1.0.

[7] Coffea. https://github.com/CoffeaTeam/coffea.

[8] Scikit-learn. https://scikit-learn.org/.

[9] Tensorflow. https://www.tensorflow.org/.

[10] Pyroot. https://root.cern/manual/python/.

[11] Jupyter. https://jupyter.org/.

[12] Conda. https://docs.conda.io/en/latest/.

[13] Pip. https://pypi.org/project/pip/.

[14] Repo2docker. https://github.com/jupyterhub/repo2docker.

[15] Binder. https://mybinder.org/.

[16] Work queue. https://cctools.readthedocs.io/en/latest/work_queue/.

[17] Dask. https://dask.org/.

[18] Parsl. https://parsl-project.org/.

[19] Funcx. https://funcx.readthedocs.io/en/latest/.