Interactive Supercomputing with Jupyter

Rollin Thomas[1] and Shreyas Cholia[1]

[1]Affiliation not available

February 2, 2021

Abstract

Rich user interfaces like Jupyter have the potential to make interacting with a supercomputer easier and more productive, consequently attracting new kinds of users and helping to expand the application of supercomputing to new science domains. For the scientist-user, the ideal rich user interface delivers a familiar, responsive, introspective, modular, and customizable platform upon which to build, run, capture, document, re-run, and share analysis workflows. From the provider or system administrator perspective, such a platform would also be easy to configure, deploy securely, update, customize, and support. Jupyter checks most if not all of these boxes. But from the perspective of leadership computing organizations that provide supercomputing power to users, such a platform should also make the unique features of a supercomputer center more accessible to users and more composable with high performance computing (HPC) workflows. Project Jupyter's core design philosophy of extensibility, abstraction, and agnostic deployment, has allowed HPC centers like NERSC to bring in advanced supercomputing capabilities that can extend the interactive notebook environment. This has enabled a rich scientific discovery platform, particularly for experimental facility data analysis and machine learning problems.

Introduction

The National Energy Research Scientific Computing Center (NERSC), located at Berkeley Lab, is the principal scientific computing facility for the US Department of Energy (DOE) Office of Science. More than 7,000 scientists, including many early-career scientists and students, use NERSC for basic non-classified research that involves supercomputing and massive data: modeling Earth's climate, studying properties of materials, probing the evolution of the Universe, analyzing data from collisions of subatomic particles, understanding protein structures, and more.

Evolution in NERSC's workload over the past 5 years has been driven by the explosion of data in all of these science areas. Experimental and observational science facilities (microscopes, telescopes, genome sequencers, particle accelerators) increasingly need leadership computing to make sense of all the data they generate. Artificial intelligence (AI) models designed to solve complex problems are trained using both massive data sets and supercomputer simulations. Jupyter has become the *de facto* platform for data science and AI, so it seems natural for data-intensive science users to expect Jupyter to work on a supercomputer.

In 2016, NERSC began exploring whether we could rely on Jupyter to provide a rich user interface to the new *Cori* supercomputer alongside traditional command-line interface (CLI) and remote desktop (i.e., NX) access modes. Today around 700 unique users per month are using Jupyter on Cori, a figure that has tripled over the last three years. In a typical month about 3,000 unique users login into Cori via CLI or NX, so in some sense, 20-25% of user interaction with Cori now goes through Jupyter. Jupyter has become indispensable, a primary point of entry to Cori and other systems for a substantial fraction of all NERSC users.

How It Started and How It's Going

In 2015, we observed with increasing regularity that users were trying to use SSH tunnels to launch and connect to their own Jupyter notebooks on *Edison*, a previous generation supercomputer. One user even published a blog post about how to do it. NERSC recognized that Jupyter notebooks and similar tools were a part of the emerging data science ecosystem we would need to engage, understand, and support. Faced with the challenge of how we would authenticate users and launch, manage, and proxy their notebooks, we began discussions with our colleagues at U.C. Berkeley (Fernando Pérez, Min Ragan-Kelly, and others) who were growing and expanding the Jupyter ecosystem to include institutional deployments. This led us to JupyterHub, which had been released a few months earlier, to address just those issues. JupyterHub provides a managed multi-user Jupyter service to enable access to computational environments and resources. It has a highly extensible, deployment-agnostic design built on powerful high-level abstractions (spawners, authenticators, services), and is developed by a robust, broad open-source community. From the perspective of an organization taking on the challenge of supporting any platform for a diverse and demanding user base, these characteristics represent potential strategic leverage.

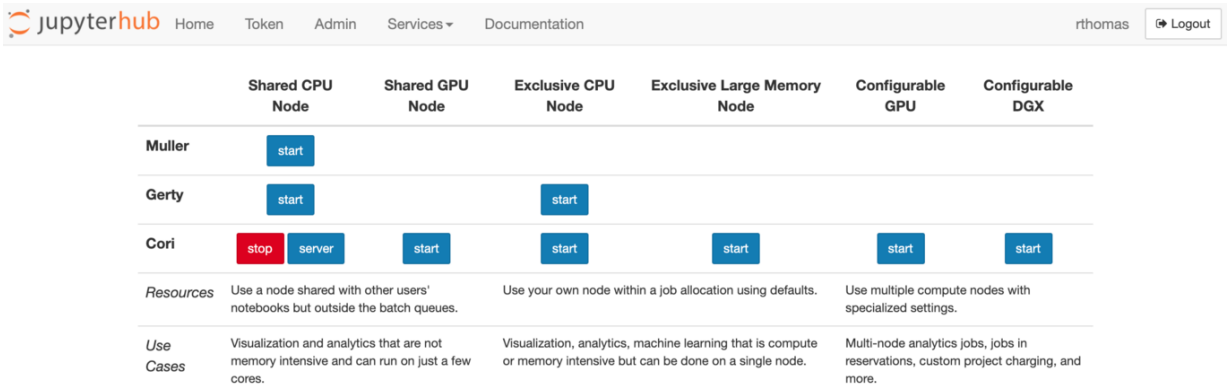

Figure 1: *NERSC's JupyterHub provides a single point of entry to running Jupyter on a variety of HPC systems with different configurations.*

Our deployment has proceeded in overlapping phases with sometimes parallel configurations for different use cases:

Phase 1: JupyterHub as science gateway. First, Jupyter provided users with the ability to run notebooks and kernels that could access data on NERSC's Global Filesystem (NGF), a collection of massive file systems for project- or group-level storage and user home directories. The principal use cases were smaller-scale analytics and visualization of data, limited by hardware resources. This phase leveraged repurposed hardware external to the supercomputers and so could not provide access to platform storage or interact with batch queues. But it was a start that signaled to users that Jupyter use was welcome.

Phase 2: Jupyter on a Cori login node. Through a custom JupyterHub spawner based on SSH, we gave users a means to launch notebooks on Cori from an external hub. These notebooks could access Cori platform storage, interact with batch queues, run Jupyter kernels based on user-customized containers, and more. Networking adjustments enabled direct connections from login nodes to compute nodes for persistent notebooks to interact with analytics (e.g., Spark, Dask) or visualization (e.g., yt) applications. This all helped make Cori a much more data-science friendly supercomputer. Phases 1 and 2 overlapped, and for some period of time two hubs were managed in parallel on reclaimed servers, a difficult-to-maintain situation that also confused users.

Phase 3: Jupyter as interface to an HPC center. Moving JupyterHub to a Docker container-as-a-

service platform at NERSC based on Rancher helped us address these issues, increased stability, and provided a platform for continuous enhancement. Extending JupyterHub's named-server support and leveraging community projects like wrapspawner and batchspawner has given us a paradigm for systematically expanding access to more shared nodes and compute nodes on Cori, but also for integrating new hardware upon delivery: New supercomputers, smaller special-purpose clusters (e.g., GPU clusters), and staff-only test systems. JupyterHub now serves as a single point of entry to Jupyter at NERSC for all users and staff (Figure 1).

Phase 4: JupyterLab as innovation platform. JupyterLab, introduced in 2018, addresses certain challenges that various software workflows face with notebooks alone. It enhances the Jupyter user experience through a powerful, modular extension framework that enables text editors, terminals, file viewers for various data formats, other custom components side-by-side with the notebook in a tabbed workspace. We have leveraged this framework to preserve core Jupyter user experience and functionality while providing enhancements that expose HPC center features in a more user-friendly way. Example of integrations and tools arising from our research partnership:

- **File system navigation:** HPC centers have large shared file systems with complex, sprawling folder hierarchies. JupyterLab users at NERSC often "got lost" while navigating NGF via UI. To effectively manage points of entry relevant to each user, we developed the jupyterlab-favorites and jupyterlab-recents *labextensions* to bookmark or highlight commonly used files and directories (e.g., scratch directories, shared project directories).
- **Batch queue:** NERSC uses the Slurm Workload Manager to manage batch job allocations on Cori's compute nodes. We have created jupyterlab-slurm, a labextension that lets users monitor and submit batch jobs directly from JupyterLab, creating a common interface to the interactive and batch components of a supercomputing workflow. We are also developing integrations for the Dask distributed framework to scale out notebook code on a set of worker nodes.
- **Reusability and reproducibility:** In team science with Jupyter it is common for a user to develop a recipe or template notebook that captures a scientific workflow, along with its software dependencies in a Jupyter kernel, and then share this with collaborators. We have developed the clonenotebooks *serverextension* to allow users to clone and run such notebooks and kernels from an nbviewer gallery. Users in a collaboration can then reproduce a given workflow and extend it to handle new data sets and parameters. Creating a general pattern of reproducible Jupyter HPC workflows with containers similar to Binder [1] is a topic of current work.

The early phases were organic with lessons learned and re-learned as we engaged the Jupyter ecosystem and the ecosystem itself matured. From this experience, we have come to appreciate the strategic value of Project Jupyter's commitments to abstraction, extensibility, and deployment-agnostic design.

Jupyter + HPC = Science!

These innovations can best be illustrated through a set of real-world science use cases that leverage our Jupyter HPC work.

Geophysical Subsurface Imaging

This use case involves running geophysical simulations and inversions for subsurface imaging. 1000 1D inversions were run to each produce a layered model of subsurface conductivity; these were then stitched together to create a 3D model to understand why the Murray River in Australia was becoming more saline. Jupyter was used to combine simulations, data analysis, and machine learning with interactive visualization. The initial workflow was developed on the user's laptop environment and needed to be scaled up at NERSC.

Jupyter was run in a Docker container at NERSC with a pre-defined reproducible software environment. Parallel computing workers were launched on Cori from Jupyter with Dask-jobqueue. Workers could be scaled up or down on-demand. Jupyter farmed out parallel tasks to Dask; the results of these parallel runs

were pulled back into the notebook and visualized. A large batch of simulations was run to generate data for a machine learning application.

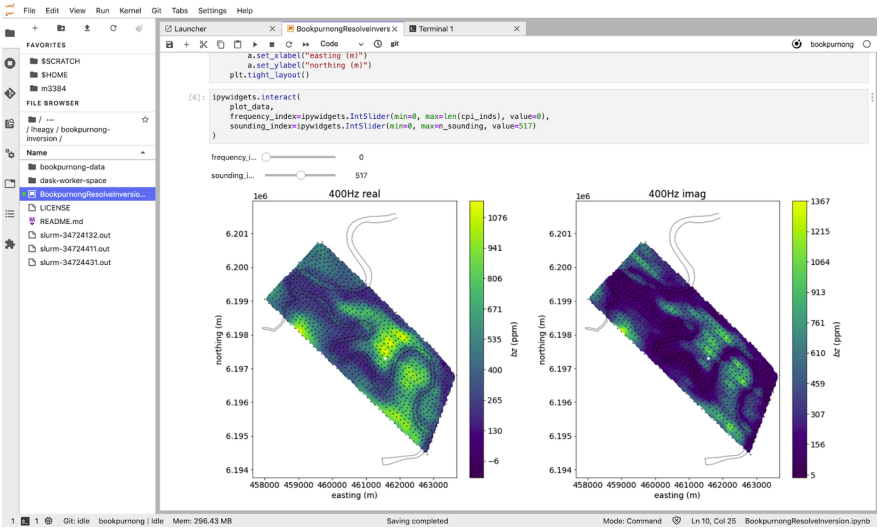

Figure 2: Geophysics Inversion Notebook generating a 3D model using NERSC HPC resources

Several other projects make use of similar patterns including the National Center for Electron Microscopy (NCEM) and the Advanced Light Source (ALS) [2] where image data are collected at an experimental facility, and then processed and analyzed with these types of Jupyter-based workflows.

Electron Microscope Image Analysis

Scanning transmission electron microscopy (STEM) is used for spectroscopy, diffraction, and imaging of materials. It is possible to determine many structural properties of materials using 4-dimensional STEM images. "Bragg Disk" detection is an important part of this process and can require computationally expensive image-processing steps.

Scientists at NCEM originally developed a serial version of the Bragg Disk detection algorithm using Python and NumPy, running as a Jupyter notebook, where electron microscope image data were rendered for inspection and analysis. The image processing could be easily vectorized and was a good candidate for parallelization with a tool like Dask. By enabling a parallel implementation for the Bragg Disk detection code through Jupyter at NERSC, we were able to significantly improve run times for large data: processing a 300GB dataset went from days to a few minutes. This workflow was scaled up to run on 40 Cori nodes (1280 workers), with results collected and visualized in the Jupyter Notebook.

This type of scaling has opened the door for computational analyses that were previously impractical. One of the benefits of the 4D-STEM technique is the ability to perform multi-scale data analysis, combining atomic-scale spatial sampling with very large fields of view. Without a scaled parallel implementation, one would need to give up either the spatial sampling or the field-of-view when analyzing larger datasets.

Advanced Light Source Tomography

Reproducibility and repeatability are very important for tomographic analyses at the ALS, where users need to repeat an experimental run on their own data. Scientific facility users may have limited programming skills, and often just need a pre-defined computational analysis that they can run and modify without having to develop the code from scratch. Jupyter notebooks are a great way to enable this type of user: analysis

notebooks become recipes that can be minimally tweaked to run with different parameters or data. In our work with the ALS, we enabled a common, shared software environment where users could run a set of curated notebooks managed by the project [3]. The specialized software environment included tomography software, along with custom Jupyter widgets to support interactive path selection and viewing of sinograms or digital slices through 3D data - installing and maintaining this can be challenging for end users. Our tool, clonenotebooks (based on nbviewer), enabled browsing of Jupyter notebooks in repositories such as GitHub, and cloning them over to NERSC with a pointer to this managed execution environment.

Exploring parameter spaces is important in tomographic analysis, where one may need to experiment with different combinations of parameters to determine the best fit for the data. Papermill (developed at Netflix) allow us to execute parameterized Jupyter notebooks, where certain variables can be replaced across runs. We developed a Papermill-based workflow to enable interactive exploration of parameter spaces for ALS users, where each parameter set can be applied to a common template notebook. Once a notebook with an optimal parameter set is identified, this can then be applied to additional data sets. HPC scaling is achieved through Dask, similarly to the earlier use cases.

Conclusion

Jupyter is proving to have a transformative impact on modern computational science by enabling a new integrative mode of interactive supercomputing, where code, analysis, and data all come together under a single visual interface that can seamlessly access powerful hardware resources.

The patterns illustrated in the above use cases are very flexible and can be applied to other science domains as well. We see common themes around parallel execution, reproducible environments and interactive visualization come up repeatedly in our work enabling Jupyter for science on HPC systems.

Jupyter is quickly becoming the entry point to HPC for a growing class of users. The ability to provision different resources, and integrate with HPC workload management systems through JupyterHub is an important enabler of easy-to-use interactive supercomputing.

Because of mission, design, and technological trends, supercomputers and the HPC centers that run them are still less homogeneous as a group than cloud providers. This means "one size fits all" solutions are sometimes harder to come by. And while providers of supercomputing power want to increase ease of use, they are not interested in homogenizing or concealing specialized capabilities from expert users. Developers working on Jupyter projects that may intersect with HPC especially should avoid making assumptions about HPC center policy (e.g., queue configuration, submit and run limits, privileged access) and seek input from HPC developers on how to generalize those assumptions. As long as Jupyter developers remain committed to extensibility, abstraction, and remaining agnostic about deployment options, developers at HPC centers and their research partners can help fill the gaps.

There seems to be considerable momentum around Jupyter and HPC. We have built a network of contacts at other HPC centers to collaborate with and learn from. In 2019, with the Berkeley Institute for Data Science, we hosted a Jupyter Community Workshop on Jupyter at HPC and research facilities to kickstart that process.

At NERSC our next step is to renew focus on supporting Jupyter at new scales and on new hardware. While we do a good job of meeting the needs of several hundred users per month, we need to do more to ease the barriers to entry to analytics cluster software and parallelized visualization tools. We look forward to this work on the Perlmutter machine beginning this year.

Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231. We would like to thank the core Jupyter team, and especially our collaborators at U.C. Berkeley for technical guidance and support around the Jupyter and JupyterHub ecosystem.

References

[1]S. Cholia *et al.*, "Towards Interactive, Reproducible Analytics at Scale on HPC Systems", *IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, 2020.

[2]M. Henderson, W. Krinsman, S. Cholia, R. Thomas, and T. Slaton, "Accelerating Experimental Science Using Jupyter and NERSC HPC", in *Communications in Computer and Information Science*, Springer International Publishing, 2020, pp. 145–163.

[3]D. Parkinson, H. Krishnan, D. Ushizima, M. Henderson, and S. Cholia, "Interactive Parallel Workflows for Synchrotron Tomography", in *XLOOP 2020*, 2020.