Towards Interactive, Reproducible Analytics at Scale on HPC Systems

Shreyas Cholia
Lawrence Berkeley National Laboratory
Berkeley, USA
scholia@lbl.gov

Lindsey Heagy
University Of California, Berkeley
Berkeley, USA
lheagy@berkeley.edu

Matthew Henderson
Lawrence Berkeley National Laboratory
Berkeley, USA
mhenderson@lbl.gov

Drew Paine
Lawrence Berkeley National Laboratory
Berkeley, USA
pained@lbl.gov

Jon Hays
University Of California, Berkeley
Berkeley, USA
jthays@berkeley.edu

Ludovico Bianchi
Lawrence Berkeley National Laboratory
Berkeley, USA
lbianchi@lbl.gov

Devarshi Ghoshal
Lawrence Berkeley National Laboratory
Berkeley, USA
dghoshal@lbl.gov

Fernando Pérez
University Of California, Berkeley
Berkeley, USA
fernando.perez@berkeley.edu

Lavanya Ramakrishnan
Lawrence Berkeley National Laboratory
Berkeley, USA
lramakrishnan@lbl.gov

***Abstract*—The growth in scientific data volumes has resulted in a need to scale up processing and analysis pipelines using High Performance Computing (HPC) systems. These workflows need interactive, reproducible analytics at scale. The Jupyter platform provides core capabilities for interactivity but was not designed for HPC systems. In this paper, we outline our efforts that bring together core technologies based on the Jupyter Platform to create interactive, reproducible analytics at scale on HPC systems. Our work is grounded in a real world science use case - applying geophysical simulations and inversions for imaging the subsurface. Our core platform addresses three key areas of the scientific analysis workflow - reproducibility, scalability, and interactivity. We describe our implemention of a system, using Binder, Science Capsule, and Dask software. We demonstrate the use of this software to run our use case and interactively visualize real-time streams of HDF5 data.**

***Index Terms*—Interactive HPC, Jupyter, Containers, Reproducible Science**

I. Introduction

Scientific data is increasingly processed and analyzed on HPC systems. These workflows are different from traditional batch-queue workflows and require more support for a unified, interactive, real-time platform that can operate on High Performance Computing (HPC) systems and very large datasets.

In recent years, scientists across a wide variety of disciplines have adopted the Jupyter Notebook as an essential element of their discovery workflows. Jupyter combines documentation, visualization, data analytics, and code into a single unit that scientists can share, modify, and even publish. The Jupyter platform and ecosystem is increasingly being deployed on clusters and HPC systems to support scientific workflows. However, there are gaps in the current Jupyter ecosystem that need to be addressed to enable interactive and reproducible analytics at scale. Our work addresses these gaps. There are three key components driving our work:

- **Reproducible Analytics:** Reproducibility is a key component of the scientific workflow. We wish to to capture the entire software stack that goes into a scientific analysis, along with the code for the analysis itself, so that this can then be re-run anywhere. In particular it is important to be able reproduce workflows on HPC systems, against very large data sets and computational resources. Further, we also need to capture any updates made to the analysis on subsequent runs.
- **Scalable Analytics:** In order to run large scale interactive workflows on HPC systems, we need to engineer these interactive analyses to utilize multiple nodes, in a distributed fashion. We need to be able to interface with HPC batch queues and launch a set of workers that can enable interactive analytics at scale.
- **Interactive Analytics:** Data analytics tends to be a highly interactive process. While traditionally, these have been run on desktops and stand-alone computers, the size of the data is requiring us to consider the modalities of these workflows at scale. There is an increased need to support interactive analytics with human-in-the-loop, large data visualizations and exploration at batch-oriented HPC facilities, with the goal of pulling subsets of the data into interactive visualization tools on-demand as needed.

Towards this end, we have developed a system that enables these components for interactive workflows. The Jupyter Notebook provides the core analysis platform, while the JupyterLab extension framework allows us to add custom functionality for interactive visualization. We use Binder [1], [2] and Science

Capsule [3] to capture a reproducible software environment. Dask [4] (an open source framework and library for distributed computing in Python) provides us with a backend to scale the analyses on the National Energy Research Scientific Computing Center (NERSC) compute nodes. Finally, we developed a JupyterLab extension to allow users to interactively work with large volumes of data. Our system allows users to capture a project specific set of Notebooks and software environment as a Git repo, launches this as containers on HPC resources at scale, and enables on-demand visualization to sub-select and view data in real-time.

II. Background

In this section, we describe Jupyter at NERSC, our science use case, and our motivating user study.

A. Jupyter at NERSC

NERSC is the primary scientific computing facility for the Office of Science in the U.S. Department of Energy. More than 7,000 scientists use NERSC to perform basic scientific research across a wide range of disciplines, including climate modeling, research into new materials, simulations of the early universe, analysis of data from high energy physics experiments, investigations of protein structure, and a host of other scientific endeavors.

NERSC runs an instance of JupyterHub - a managed multi-user Jupyter deployment. The hub allows users to authenticate to the service, and provides the ability to spawn Jupyter Notebooks on different types of backend resources.

B. Use case

We motivate our work through a real-world science use case involving geophysical simulations and inversions for imaging the subsurface. Geophysical imaging is used for resource exploration in hydrocarbons, minerals, and groundwater, as well as geotechnical and environmental applications. Subsurface variations in physical properties such as density or electrical conductivity can be diagnostic of geologic units or sought-after natural resources. Geophysical data, such as gravity or electromagnetic measurements, which are sensitive to these physical properties, can be collected in the field and interpreted using computational methods.

The computational building blocks that we require for working with geophysical data include:

- The ability to simulate the governing partial differential equations (PDEs). These are typically large-scale, require parallel computation, and can generate large volumes of data that are stored to disk for later analysis.
- The ability to run PDE constrained optimization to estimate a model of the subsurface given geophysical data. This requires many simulations to be run as well as mechanisms for estimating derivatives in the optimization.

More recently, machine learning (ML) techniques are being used in physically-motivated scientific contexts, both as companions/replacements for parts of traditional computational pipelines (e.g. playing the role of effective models or accelerated solvers) and as tools capable of integrating multimodal data for prediction, beyond what traditional physical models could do. In the geosciences we have access to both of these use cases, and the current project focuses on using ML tools to develop accelerated models that can accurately represent fine-scale physics that is computationally expensive to solve for directly.

This requires generating a substantial amount of training data from first-principles solvers, and then training the ML pipelines on the data to capture the relevant physical features at the required accuracy. This new ML-based effective model then needs to be used in the complete pipeline, in this case, opening the possibility of solving more physically realistic problems in finite time thanks to this accelerated component.

This process is both data- and computationally-intensive (and thus well suited for an HPC environment) but also requires expert insight and evaluation throughout. It is critical that the scientist with domain knowledge inspect the structure of the solutions learned and produced by the ML system to ensure that, in addition to the constraints encoded in the underlying optimization problem, they have the right overall physical structure. In the training process, visual and interactive exploration of the complex data is necessary to understand where issues may arise, as summary metrics such as total loss are often insufficient to inform the scientist.

Since the simulation codes tend to be optimized in their output formats and structure for computational efficiency, they do not always automatically produce the variables most convenient for interactive exploration. Furthermore, it would be cumbersome to require such capabilities to be hard-coded in the simulations themselves, as the requirements of interactive exploration can vary from problem to problem based on multiple conditions (questions being asked, bugs/issues found in a particular run, parameter values, etc.). It is thus optimal to separate the low-level capabilities of the code that generates all the proper generic data, from the exploratory environment where a scientist can probe the specific combinations of quantities they need at a given time.

In this work, we design the right level of interactive capabilities in the JupyterLab environment to allow the working scientist to flexibly explore and query their data at multiple levels, with a minimal amount of customization required of the underlying simulation. Additionally, we tackle two issues that are part of the entire lifecycle of research and that become particularly acute in HPC contexts: how to improve the experience of interfacing with the HPC system's scheduling environment for a scientist focused on exploratory questions, and how that scientist can then best share the results of their work with others in a self-contained, reproducible manner.

C. Jupyter User Study

Our work is also motivated by a qualitative user study conducted with individuals who use Jupyter as part of their work with user facilities [5] that investigated scientific needs and requirements for Jupyter in HPC environments. We interviewed thirty-two researchers, computer scientists, and open source developers between September and December 2019. Subsequently, we surveyed recent users of NERSC's Jupyter installation between March - April 2020 (receiving 103 responses from over 900 users contacted). Combined, these datasets explore how facilities are supporting their scientific user's work with Jupyter, how scientists are using Jupyter to accomplish different research activities, and identifies gaps in the alignment of Jupyter tools and HPC facility systems and cultures.

Jupyter users at NERSC are frequently using this resource to interactively explore and analyze data housed on the different shared filesystems. A handful of individuals are prototyping analysis code and running machine learning workflows, and a few are executing batch processing jobs from Notebooks. Sharing and collaboration centered around Jupyter Notebooks also emerged as key topics in interviews and our survey. All of our interviewees indicated they share Notebooks in some way, typically with collaborators and at times the public or their community. Additionally, 69% of our survey respondents share Notebooks through one or more mechanisms including email, version controlled repositories like GitHub, a shared file system (NERSC supports shared global filesystems), and cloud based sharing services like Slack, Dropbox, or Google CoLab.

Sharing and collaborating on Notebooks is challenging for a few common reasons. First, clean version controlling these files requires using extra tools, like nbdime [6], that some users are unaware of. Second, there is a need to share the computational environment setup if a collaborator is going to execute a shared Notebook on their own system. Related to both of these issues, participants noted their work often requires large datasets housed on shared filesystems. When sharing Notebooks their colleagues require access to the data to run and iterate on the work with its artifacts. When a colleague doesn't have access to the HPC system hosting the data they either cannot fully use a shared Notebook, have to obtain access to the system, or have to figure out how to adapt and use a subset of the data. These challenges help motivate our work to build a reproducible, shareable, and scalable interactive data analysis pipeline using Jupyter tools, the Science Capsule framework, and NERSC resources.

III. Architecture

Figure 1 shows the prototype architecture we have developed that would let users start from a Git software repository to create an interactive, shareable, and reproducible analysis environment at NERSC. A user can use their locally cloned Git repo along with a Science Capsule Docker environment and the repo2Docker [7] tool to create a Docker [8] image that can then be registered in a container registry. Subsequently, this image is converted to a Shifter [9] image. Shifter is a container technology that allows user-created Docker images to run at NERSC. A user can come to NERSC and use JupyterHub to launch the created image at NERSC. Using our tools, a user is able to deploy their Notebooks and environment from their desktop to an HPC machine.

A. Binder on HPC

Maintaining a consistent software environment to run a given analysis presents a hidden cost to the scientist. Software dependency management is a potentially time-consuming task, and adds roadblocks as the scientist must grapple with installing and managing a working software tool chain with the analysis itself. It hinders reproducibility since one needs to have a consistent software environment to be able to reproduce a given analysis.

The Binder Project in the Jupyter ecosystem provides a framework to capture software specifications starting from a Git repo, converting them into containers and deploying these into live computing environments. With Binder, users can place Notebooks along with a dependency file in a repository, and then simply share a link to the repository with others to allow them to run the Notebook. Binder is designed to run in a cloud environment with built-in containerization support for Docker and Kubernetes [10]. However, it does not work natively on traditional HPC systems.

Our goal has been to create a reproducible Notebook in an HPC setting, such that we can run workflows against large datasets that are being generated from experimental workflows or simulations. We are looking at taking Binder's container based approach and applying it to reproducible environments at HPC centers like NERSC. Our solution is based on adapting the Binder toolset to HPC, using *Shifter* to allow user-created Docker container images to be run at NERSC, along with the *Science Capsule* software to capture provenance within the container (described in Section 3.3).

In our current implementation, users store their analysis Notebook in a remote Git repo (we use GitHub in our work), and a software specification that goes along with it. Binder's repo2docker tool provides a standard set of protocols to convert such a repo into a Docker container at the click of a button.

We create an initial build of a Docker container that includes the: a) software environment needed to run the Notebook code b) underlying Git repo with the analysis notebooks and related code, c) Science Capsule software to capture provenance for reproducibility in a running environment, and d) Dask software to scale up the analysis. This container is then pushed to the NERSC Docker image registry.

Once an image is available to run we pull it from the registry onto the HPC system (Cori) and convert it into a Shifter image, so that it can run on NERSC systems. Shifter is NERSC's container implementation that allows Docker containers to run on HPC nodes, and is similar to other HPC container tools like Singularity [11] or CharlieCloud [12].

We use the NERSC Jupyterhub service to launch the software environment. Jupyterhub is a service that provides an institutional multi-user deployment of Jupyter Notebook services. NERSC allows users to launch custom versions of their Jupyter environment via Jupyterhub. In this case, we

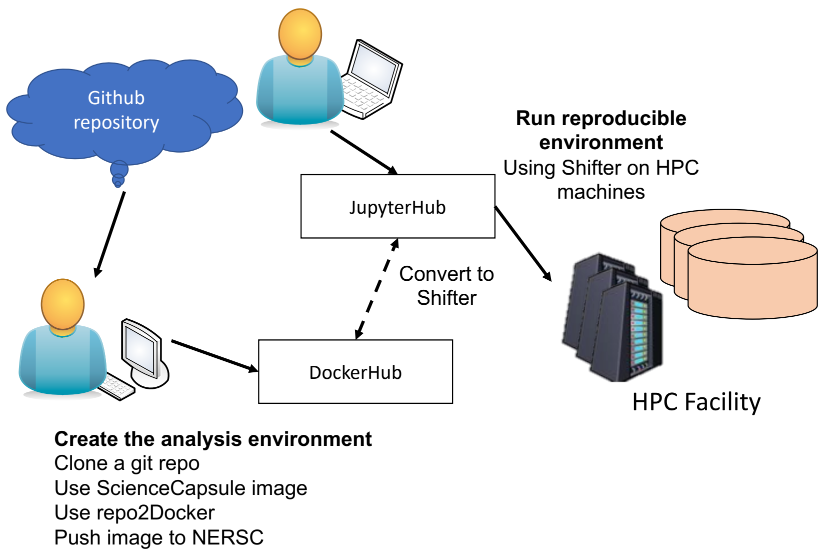


Fig. 1. The figure shows our prototype that allows users to build a reproducible environment to share. A user can start from a Git repository and an image to create a customized analysis environment which can be pushed to NERSC. The registered image is then accessible to users to launch at NERSC.

point the NERSC Jupyter environment at the Shifter image that was just built. This lets us launch this container on the Cori supercomputer, and places the user into a live interactive Jupyter environment.

We note that while this work was prototyped on NERSC systems using a specific technology stack, we believe that this overall approach is technology-agnostic. The Binder for HPC approach should work for any container technology compatible with Docker - it would involve replacing the Shifter container run-time with an alternate engine (e.g., Singularity or CharlieCloud), and converting the image to the desired run-time on the HPC side. Using containerization ensures that the process for packaging and building these tools remains the same across different HPC backends.

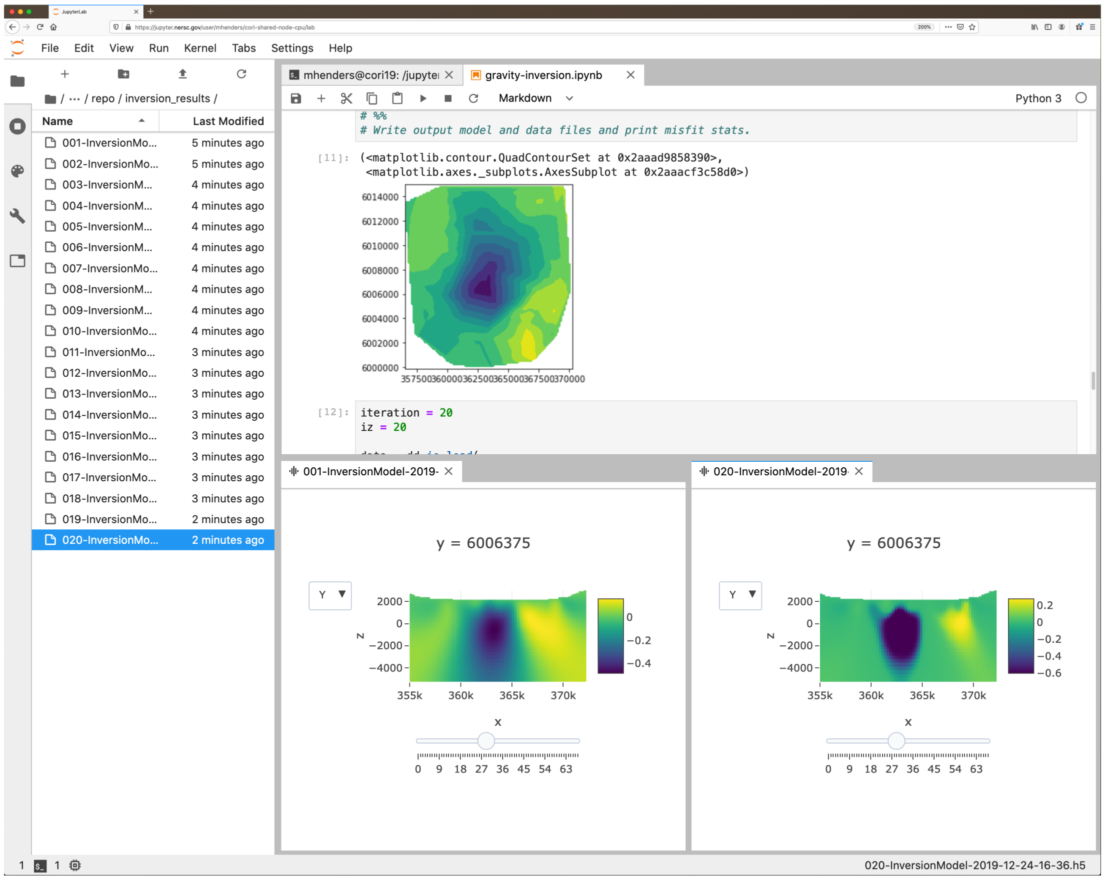
B. Integration with Dask

Enabling our reproducible containers described in the previous section to run at scale on HPC resources presents a new set of challenges and opportunities. We would like to enable HPC users to easily share Notebooks that can be re-run, including executing large-scale applications. This means creating a seamless environment to allow existing Notebook-based workflows to easily grow to larger scales across very large datasets with minimal effort on the users end.

In order to scale up computations from a Jupyter Notebook, it needs to interact with the HPC backend to utilize additional compute resources. We use the Dask framework to manage and launch interactive workers on HPC compute nodes.

In previous work, we developed a Dask-based backend for the Jupyter interface [13], [14], where we demonstrated the scale up of image processing for 4D STEM electron microscope data across 1920 cores (achieving 100x to 600x performance improvements). In this paper, we build on our earlier work by adding Dask as a layer to enable scale-up of our reproducible environments described in section 3.1. Dask is included as a dependency in the container that we build with Binder. This allows us to instantiate Dask from within this container. We extend the Jupyter Notebook to use Dask to leverage performance at scale on HPC systems — using its advanced parallelism capabilities for Python based analytics, to provide hooks to directly execute code in the Notebook remotely. Dask workers enable us to farm out and vectorize Notebook cells or functions. We can instruct the Notebook to run selected code on remote Dask workers, and collect results asynchronously back into the Notebook process, so that they can be analyzed and visualized in the Notebook. This gives us a powerful and simple mechanism to scale-up a piece of interactive code on an HPC backend.

NERSC uses the Slurm batch scheduling system to manage jobs on the Cori supercomputer. Using Dask's Slurm Job Queue integration, we were able to launch a set of workers that were launched on Cori compute nodes through Slurm. In collaboration with NERSC, we were able to determine the appropriate set of queue characteristics to enable scalable, interactive analytics. In particular it is very useful to have a queue that offers nodes on-demand or with a short wait time, since the user is expected to work interactively with the analysis. To maintain these characteristic the queues needed to be over-provisioned (so that free nodes were always available), with limits on the maximum job size (so that a single job couldn't use up all the resources), along with a very high priority boost (so that these jobs were always preferentially run before others). NERSC provides interactive and real-time queues that have a very short turnaround time, and are suitable for these kinds of workloads. In future work, we will be investigating ways to increase the efficiency of



Name	Last Modified
001-InversionMo...	5 minutes ago
002-InversionMo...	5 minutes ago
003-InversionM...	4 minutes ago
004-InversionM...	4 minutes ago
005-InversionM...	4 minutes ago
006-InversionM...	4 minutes ago
007-InversionMo...	4 minutes ago
008-InversionM...	4 minutes ago
009-InversionM...	4 minutes ago
010-InversionMo...	4 minutes ago
011-InversionMo...	3 minutes ago
012-InversionMo...	3 minutes ago
013-InversionMo...	3 minutes ago
014-InversionMo...	3 minutes ago
015-InversionMo...	3 minutes ago
016-InversionMo...	3 minutes ago
017-InversionMo...	3 minutes ago
018-InversionMo...	3 minutes ago
019-InversionMo...	2 minutes ago
020-InversionMo...	2 minutes ago

```
# %%
# Write output model and data files and print misfit stats.

[11]: (<matplotlib.contour.QuadContourSet at 0x2aaad9858390>,
       <matplotlib.axes._subplots.AxesSubplot at 0x2aaacf3c58d0>)

[12]: iteration = 20
      iz = 20
```

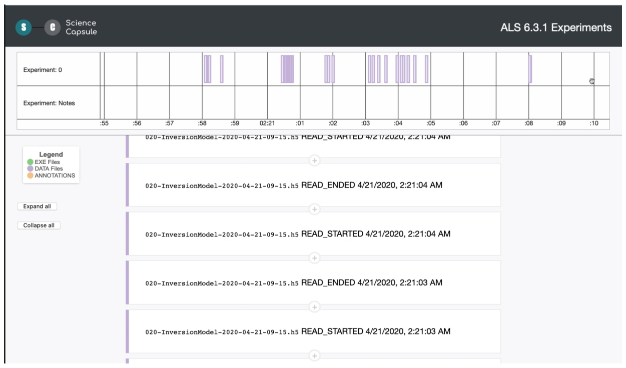
Fig. 2. JupyterLab HDF5 Slicer prototype that allows users to interactively explore 3D models simply by double-clicking the appropriately formatted HDF5 files.

provisioning Dask workers in an HPC environment, since Dask Job Queue launches individual jobs for each worker, rather than submitting a set of workers in bulk.

In our example, we worked with the SimPEG Geophysics code, and were able to execute simulation runs remotely on 40 Dask workers using this approach. This enabled us to streamline the setup for running of a large batch of simulations used to generate data for a machine learning application. Similarly, many other geophysical problems, such as frequency domain electromagnetics, are readily parallelizable, enabling us to also improve run-times of individual simulations.

C. Science Capsule environment

Science Capsule is a framework that helps scientists capture, curate, and revisit artifacts produced and used over the lifecycle of their work. This framework is run in a Docker container, or bare-metal on a machine, and monitors a user's specified computational workflows to automatically capture events from the run-time environment. These events are then used to generate detailed provenance information for future reproduction of the work. Importantly, Science Capsule enables scientists to incorporate their paper and digital notes alongside this provenance information so that they can explore and revisit their work in a web-based timeline (Figure 3). The provenance generated by Science Capsule helps users make their analyses more reproducible.



```
020-InversionModel-2020-04-21-09-15.h5 READ_STARTED 4/21/2020, 2:21:04 AM
020-InversionModel-2020-04-21-09-15.h5 READ_ENDED 4/21/2020, 2:21:04 AM
020-InversionModel-2020-04-21-09-15.h5 READ_STARTED 4/21/2020, 2:21:04 AM
020-InversionModel-2020-04-21-09-15.h5 READ_ENDED 4/21/2020, 2:21:03 AM
020-InversionModel-2020-04-21-09-15.h5 READ_STARTED 4/21/2020, 2:21:03 AM
```

Fig. 3. The Science Capsule user interface displays a timeline of events captured while allowing users to add notes and images for later use when revisiting their work.

D. Rich, interactive visualization of HDF data

Interactive visualization is a key element of the scientific analysis loop and allows the user a means to query and inspect data and results from computational runs. Jupyter fits naturally into this space, as it allows users to make use of a vast ecosystem of front-end JavaScript data visualization tools that can run in the browser, while connecting these with the backend Python machinery to interrogate the data. In the context of HPC and big data, this presents a unique set of challenges, because the datasets can be very large; it may not be practical or feasible to pull in the entire dataset into a remote client browser for visualization.

In order to address this we developed a prototype JupyterLab extension that visualizes structured data in the Hierarchical Data Format version 5 (HDF5) format [15]. HDF5 is a common format for representing very large, multi-dimensional arrays of structured data. Our extension, *the JupyterLab HDF5 slicer* [16] allows users to sub-select and query data from a multi-dimensional array in an HDF5 file using an HTTP API.

In our geophysics use case, the data that are produced describe a 3D model of the subsurface at each iteration of an optimization problem. These results are recorded to disk using the HDF5 format. Being able to quickly preview these results, during execution, enables users to assess the setup, choice of hyper-parameters, and progress of the optimization problem; this can be important for identifying and isolating problems during the computation. Also, the ability to scan through two-dimensional slices of results provides a way to compare, visualize, and interpret 3D results in the context of the geologic question being asked. The data is remote to the user, in files residing on the HPC filesystem, while the user visualizes data in JupyterLab from a web browser in their local environment (e.g., laptop or desktop).

The extension allows a user to simply double-click on an appropriately formatted HDF5 file and visualize the data, which is shown in Figure 2. Data is fetched on-demand for rapid visualization and to minimize unnecessary remote data transfer for data not viewed, which could potentially overwhelm the network and the browser. There is a server side component that is launched through the Jupyter web service backend (Tornado [17]) that can query HDF5 data directly through the h5py library, and exposes this via an HTTP API.

In conjunction with the JupyterLab HDF5 extension [18], our viewer provides a tool for rendering 2-dimensional orthographic views of 3-dimensional data, allowing users to interactively slide through individual slices of data that are pulled down from the server and rendered in real time.

Three interactive controls are provided: (1) normal axis selector: a drop-down menu which determines which axis is perpendicular to the screen (i.e., the dimension which is sliced across); (2) slice selector: a slider which determines the current slice index and allows the user to traverse the normal axis; (3) color-scale selector: a drop-down menu which determines the color scheme of the heat map. These controls enable us to manage the data streaming to the interface from the backend API.

IV. Related Work

In this section, we detail some of the work in interactive, collaborative, and reproducible analytics.

Interactive Computing and HPC Scientific workflows have been used for composing and executing pipelines on distributed and HPC platforms and provide some interactive computing capabilities [19]. There are a number of interactive computing environments including Matlab, SAS, Excel, Mathematica [20], and Apache Zeppelin [21]. There are numerous electronic lab Notebook systems (ELNs), such as Benchling [22] and SciNote [23], which serve more as laboratory information management and documentation systems. These types of ELNs are different from live computational Notebooks like Jupyter which serve as interactive live code environments that combine documentation and visualizations. The Artificial Intelligence/Machine Learning (AI/ML) space also has a number of interactive tools that specifically provide integration with popular ML libraries. Tensorboard provides a visual interactive interface for Tensorflow [24] based workflows.

There are numerous efforts at various DOE and NSF facilities to enable Jupyter in HPC. Work done on Batchspawner [25] at the University of Minnesota has enabled a lot of the initial batch system integration in JupyterHub, and is being used as the core to build a lot of the advanced resource scheduling infrastructure at NERSC. Tools like Parsl [26], Dask [4], Spark, and IPyParallel [27] manage backend workers that can be used to facilitate farming out tasks to a distributed set of nodes. We will be using one or more of these tools as appropriate.

Reproducibility and Sharing Tools like GitHub and BitBucket let users share and version control code while archives such as Zenodo [28], Dataverse [29], and Git LFS [30] provide versioned control data. However, there are limited tools that allow users to share an interactive analysis environment, especially on HPC systems. Digital libraries are a collection of data and associated metadata to support the information needs of the users [31], [32]. Previous research has proposed different ways to index and extract information from articles in digital libraries [33], [34].

National Institute of Science and Technology (NIST) has built CoRR (Cloud of Reproducible Records), a cloud infrastructure for automating interoperability and reproducibility of experiments [35]. Burrito [36] is a Linux-based system that captures a researcher's computational activities and provides user interfaces to annotate the captured provenance. Popper [37] is a protocol and command language interpreter (CLI) that uses DevOps practices to implement scientific exploration pipelines such that researchers leverage existing tools and technologies to enable reproducibility in computational experiments. Science Capsule captures and preserves the human-generated artifacts and provenance of a workflow life cycle as digital objects for knowledge discovery.

Virtualization has been used to replicate execution environments in the cloud [38]. Recent work on container technologies has enabled encapsulation of applications and associated data into a single lightweight entity. Container technologies like Docker ease the management of the life cycle of an entire application or a workflow [39], [40] and provide data encapsulation for sharing. The Large Synoptic Survey Telescope is using a custom Jupyter container based workflow [41], with a domain science software stack that can be provisioned on Kubernetes based resources, and allows scientists to reproduce analyses for this project.

The LIGO Binder effort demonstrated the feasibility and value of shareable Notebook repositories [42], [43] running on these cloud instances. Pangeo [44] is a community platform for big data geoscience, that uses a Jupyter deployment in the cloud, to run reproducible workflows through Binder using a pre-configured geoscience-specific software container, and using tools like Dask for scalability.

V. Conclusion

Our work lays the foundation for enabling reproducible, interactive analytics at scale in an HPC environment. Our work has been informed by user studies, and is grounded in a real world geophysics application. We have been able to build a prototype at NERSC demonstrating the viability of this work. We believe that this approach will be increasingly important as a means to bridge usability and interactivity with traditional HPC workflows, while providing a template for reproducible scientific analyses.

It allows us to meaningfully address interactivity and reproducibility at significantly higher scales. It allows a given analysis to be run against larger or different datasets, while maintaining an interactive real-time component. The Jupyter Notebook allows the user to go back and forth between live interactive analysis tools and large scale computations.

This ultimately enables scientific collaborations to maintain a set of shared and curated analyses and their software environments. This has the advantage that the analysis can be re-run by anyone within the collaboration without needing to configure and set up complex software requirements and dependencies. Users can fork and re-run their own versions of the analyses, while capturing new changes, thus completing the scientific workflow loop. We believe that bringing this approach to HPC provides immense value and a boost in the scope of projects that can avail of this technology.

VI. Acknowledgments

We thank Rollin Thomas for helping us with the Jupyter support at NERSC. This work and the resources at NERSC are supported by the U.S. Department of Energy, Office of Science and Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH11231.

References

[1] P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing, "Binder 2.0 - reproducible, interactive, sharable environments for science at scale," 01 2018, pp. 113–120.

[2] "Binder website," https://mybinder.org/.

[3] "Science capsule," https://sciencecapsule.lbl.gov.

[4] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, no. 130-136. Citeseer, 2015.

[5] D. Paine and L. Ramakrishnan, "Understanding interactive and reproducible computing with jupyter tools at facilities," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-2001355, 2020.

[6] "nbdime jupyter notebook diff and merge tools," https://github.com/jupyter/nbdime, 2020.

[7] P. Jupyter, "repo2docker," https://repo2docker.readthedocs.io/.

[8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[9] L. Gerhardt, W. Bhimji, S. Canon, M. Fasel, D. Jacobsen, M. Mustafa, J. Porter, and V. Tsulaia, "Shifter: Containers for hpc," *Journal of Physics: Conference Series*, vol. 898, p. 082021, 10 2017.

[10] "Kubernetes," https://kubernetes.io/docs/reference/.

[11] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, vol. 12, no. 5, 2017.

[12] R. Priedhorsky and T. Randles, "Charliecloud: Unprivileged containers for user-defined software stacks in hpc," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126925

[13] S. Cholia, M. Henderson, O. Evans, and F. Pérez, "Kale: A system for enabling human-in-the-loop interactivity in hpc workflows," 10 2018.

[14] M. L. Henderson, W. Krinsman, S. Cholia, R. Thomas, and T. Slaton, "Accelerating experimental science using jupyter and nersc hpc," in *Tools and Techniques for High Performance Computing*. Springer, 2019, pp. 145–163.

[15] T. H. Group, "Hierarchical data format, version 5," 1997–2016, http://www.hdfgroup.org/HDF5/.

[16] J. Hays, "Jupyterlab slicer," https://github.com/JonjonHays/jupyterlab-slicer.

[17] M. Dory, A. Parrish, and B. Berg, *Introduction to Tornado*. O'Reilly Media, Inc., 2012.

[18] P. Jupyter, "jupyterlab-hdf5," https://github.com/jupyterlab/jupyterlab-hdf5.

[19] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014.

[20] W. R. Inc., "Mathematica, Version 12.1," champaign, IL, 2020. [Online]. Available: https://www.wolfram.com/mathematica

[21] "Zeppelin website," https://zeppelin.apache.org/.

[22] "Benchling website," https://www.benchling.com/.

[23] "Scinote website," https://www.scinote.net/.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[25] M. B. Milligan, "Jupyter as common technology platform for interactive hpc services," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219104.3219162

[26] Y. N. Babuji, K. Chard, I. T. Foster, D. S. Katz, M. Wilde, A. Woodard, and J. M. Wozniak, "Parsl: Scalable parallel scripting in python." in *IWSG*, 2018.

[27] "ipyparallel," https://github.com/ipython/ipyparallel, 2015.

[28] M.-A. Sicilia, E. García-Barriocanal, and S. Sánchez-Alonso, "Community curation in open dataset repositories: Insights from zenodo," *Procedia Computer Science*, vol. 106, pp. 54–60, 2017.

[29] G. King, "An introduction to the dataverse network as an infrastructure for data sharing," 2007.

[30] C. Boettiger, "Managing larger data on a github repository," *Journal of Open Source Software*, vol. 3, no. 29, p. 971, 2018.

[31] C. L. Borgman, “What are digital libraries? competing visions,” *Information processing & management*, vol. 35, no. 3, 1999.

[32] G. G. Chowdhury and S. Chowdhury, *Introduction to digital libraries*. Facet publishing, 2003.

[33] S. Lawrence, C. L. Giles, and K. Bollacker, “Digital libraries and autonomous citation indexing,” *Computer*, vol. 32, no. 6, pp. 67–71, 1999.

[34] W. Michener, D. Vieglais, T. Vision, J. Kunze, P. Cruse, and G. Janée, “Dataone: data observation network for earth—preserving data and enabling innovation in the biological and environmental sciences,” *D-Lib Magazine*, vol. 17, no. 1/2, p. 12, 2011.

[35] “Corr,” https://mgi.nist.gov/cloud-reproducible-records/, 2020.

[36] P. J. Guo and M. I. Seltzer, “Burrito: Wrapping your lab notebook in computational infrastructure,” 2012.

[37] I. Jimenez, A. Arpaci-Dusseau, R. Arpaci-Dusseau, J. Lofstead, C. Maltzahn, K. Mohror, and R. Ricci, “Popperci: Automated reproducibility validation,” in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2017, pp. 450–455.

[38] B. Howe, “Virtual appliances, cloud computing, and reproducible research,” *Computing in Science & Engineering*, vol. 14, no. 4, pp. 36–41, 2012.

[39] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 171–172.

[40] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D’Souza, S. Devoid, D. a. Murphy-Olson, N. Desai *et al.*, “Skyport: container-based execution environment management for multi-cloud scientific workflows,” in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE Press, 2014, pp. 25–32.

[41] J. M. Perkel, “Why jupyter is data scientists’ computational notebook of choice,” *Nature*, vol. 563, no. 7732, pp. 145–147, 2018.

[42] B. P. Abbott, R. Abbott, T. Abbott, M. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, “Observation of gravitational waves from a binary black hole merger,” *Physical review letters*, vol. 116, no. 6, p. 061102, 2016.

[43] “Ligo collaboration open science tutorials,” https://www.gw-openscience.org/tutorials, 2015.

[44] J. Hamman, M. Rocklin, and R. Abernathy, “Pangeo: a big-data ecosystem for scalable earth system science,” in *EGU General Assembly Conference Abstracts*, vol. 20, 2018, p. 12146.