Kale: A System for Enabling Human-in-the-loop Interactivity in HPC Workflows

Shreyas Cholia*, Matthew Henderson, Oliver Evans, Fernando Pérez
Lawrence Berkeley National Laboratory,
Berkeley, USA
*scholia@lbl.gov

I. Introduction

Scientific problem-solving frequently requires interactive, iterative exploration and analysis. Web-based interactive electronic notebook interfaces such as Jupyter [1] offer an important mechanism for scientists to capture analyses in a reproducible narrative context. An increasing number of science gateway environments are providing support for Jupyter Notebooks as a means to enable custom, ad-hoc analyses on scientific data. However, Jupyter Notebooks alone are not enough to fulfill the needs of scientific researchers today. Scientists are producing and consuming large amounts of data, and require significant computational resources to process and analyze that data, causing scientific workflows to become increasingly asynchronous in nature as processing is off-loaded to remote resources. Many scientific researchers turn to HPC systems for processing, but the traditional asynchronous batch-queue environment used in HPC for such computationally intensive tasks is largely separate from interactive Notebook-based workflows, producing a fragmented workflow for scientists that does not facilitate rapid scientific inquiry. We introduce our system "Kale" that enables Jupyter Notebooks to seamlessly interface with HPC workflows, leveraging distributed computational resources for iterative human-in-the-loop scientific exploration [2][3].

II. Requirements

Our primary motivation for Kale is improved scientific discovery and productivity through better tools. We seek to enable human-in-the-loop computing, and enhance reproducibility and collaboration in the scientific HPC and big-data space. In order to contextualize our efforts, we interviewed current users at the National Energy Research Scientific Computing Center (NERSC), to understand how scientists are using Jupyter in an HPC context [8], and to identify gaps and desired features in the setup.

Key requirements that emerged:

- The need to use Notebooks for QA/QC in existing data-analysis and HPC pipelines, including auto-generated Notebooks that can be executed on results.
- Allowing human inspection of results during and after batch workflow steps.
- Scaling up single-node Notebook operations to a parallel/distributed mode
- Setting up and controlling job workflows through the Notebook interface.

Based on these requirements, our focus has been on providing a more natural development cycle for scientists using HPC through human-in-the-loop computing, with a special emphasis on real-time task monitoring, dynamic task control, and runtime ad-hoc analyses at scale. Note that we are deliberately not building a new workflow system through this effort. Rather, our focus is on creating a toolset that can easily work with with existing workflows. This includes workflows based on existing workflows tools and job managers, as well as custom ad-hoc workflows. The purpose of Kale is to provide an integration layer between interactive widgets in Jupyter notebooks and the backend workflows.

III. System Description

Our approach is to leverage the existing Jupyter architecture as much as possible, and to build upon existing pieces of the infrastructure, including Notebooks, Frontends, Widgets and Kernels. A Jupyter Notebook is a JSON document that contains a linear list of cells that are either markdown or code, and may also include rich output produced by that code, including text, image data and placeholders for interactive Widgets. A Jupyter Frontend is the web application, which allows the user to view and manipulate the Notebook, renders rich output, and handles complex user interaction with Notebook elements. Jupyter Widgets are interactive elements rendered as cell outputs by a Frontend such as sliders, text inputs, or plots that directly manipulate and react to Kernel-side variables. Finally, a Jupyter Kernel communicates with a Frontend in order to execute and introspect user code.

We are primarily interested in Jupyter Widgets and Jupyter Kernels. Jupyter Widgets are of particular interest here because they can be composed visually and logically to form powerful dashboards that enable quick, high-level exploration of data and algorithms by providing a coupled input control and output visualization interface. Jupyter Kernels are of interest for asynchronous task execution, interactive task control, monitoring, and data transfer. Future work may explore customized Kernels for tighter HPC integration.

Presented at Gateways 2018, University of Texas, Austin, TX, September 25–27, 2018.
https://gateways2018.figshare.com/

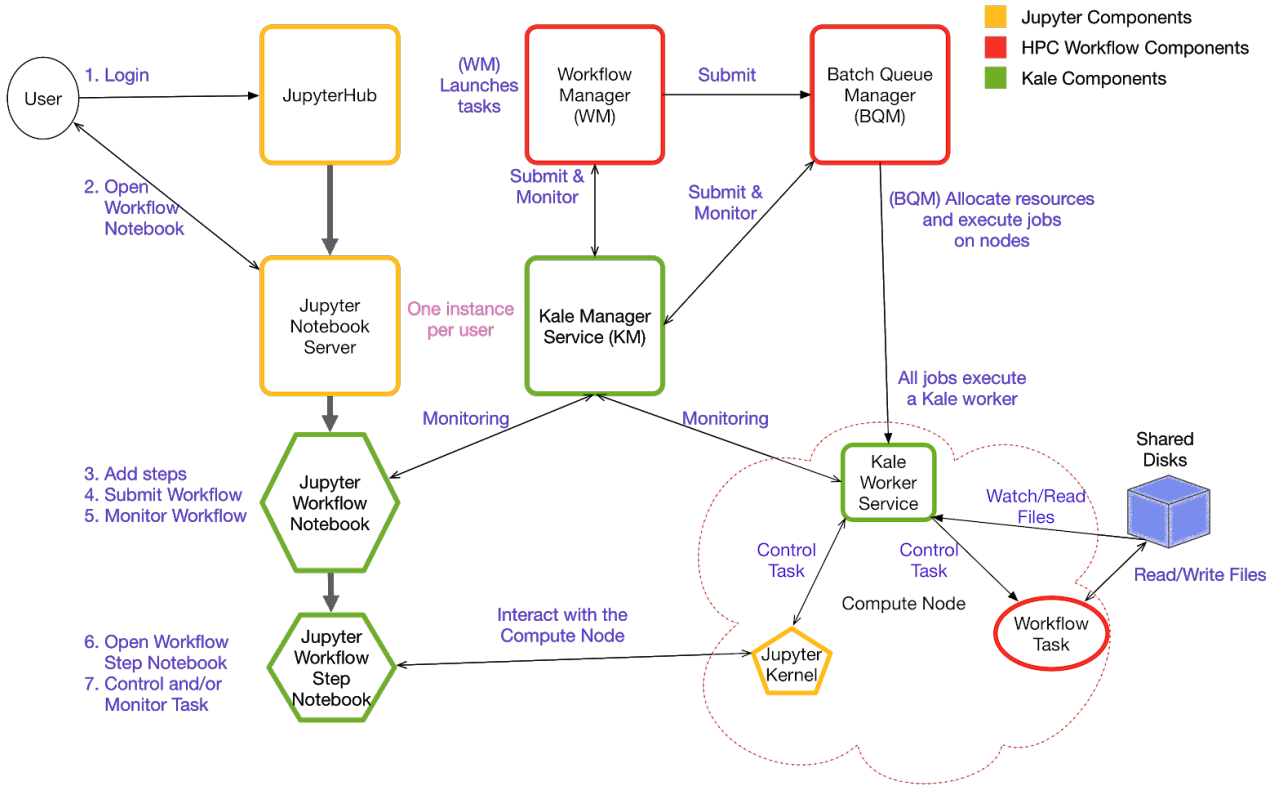

Fig. 1. High level architecture of a Jupyter deployment with Kale in an HPC environment.

We introduce from Kale two types of services. The first is a centralized Manager service that sits outside of the core Jupyter service and provides communication between a Notebook, Kale Worker services, and the HPC environment. The second type of service is a Worker service, which is used to wrap the HPC backend tasks, providing us with fine grained control over the task itself, monitoring of the job and node, and the ability to pass data back and forth. The Worker service is non-invasive to the underlying tasks, allowing existing workflows to operate without refactoring task level code.

A Kale Worker service component provides a REST API. The Notebook process can communicate with the Worker via this REST API, to give us resource monitoring at the task and node level, along with fine-grained task control including basic operations like Start, Stop, Pause, and Resume. Moreover this Worker service enables direct communication with the task and allows us to serialize results and output directly into the Notebook. We can take this a step further by using the Worker service to execute any ad-hoc operation that needs to be performed at the task or node level.

We describe the Kale architecture and how it fits into a typical Jupyter deployment in Figure 1. JupyterHub is a multi-user platform for launching Jupyter Notebook servers, and is anticipated to be the gateway Jupyter interface to an HPC system for users. Users will login through JupyterHub, taking them to their own Jupyter Notebook server instance. In tandem with the Jupyter Notebook server, a Kale Manager daemon service will be running as well. This daemon service operates as a communications broker between Jupyter Notebook Kale clients and other major system components, including the HPC Batch Queue, any Scientific Workflow managers, and individual Kale worker services running with HPC tasks. Kale assumes a Jupyter Notebook based flow of execution, using an initial 'master' Notebook capturing the overall scientific computation and analysis, with the ability to launch task level Notebooks for interactive manual runs of individual tasks. Individual task Notebooks have a Kernel running on the compute node, and a Kale Worker Service providing controlling and monitoring for each HPC task. One of the advantages of using a daemon Manager service for Kale is that Notebooks can disconnect from running HPC jobs and return later, without disrupting execution. The Kale Worker Service can also be used to monitor results from the workflow task directly. In Figure 1. we demonstrate this by reading job output from a shared filesystem, but we can also have it use more sophisticated mechanisms like communicating with a message queue or querying a remote datastore to pull results, since the actual operations can be user-defined.

IV. Using Kale With IPyWidgets

In addition to Kale's backend services, we provide modular, interactive Widget interfaces, which embed monitoring and task control functionality directly in a Notebook cell. Figure 2 shows an overview of a molecular dynamics workflow, which includes large long-running LAMMPS simulations, data parsing/analysis scripts, and results visualization in Jupyter, all as workflow tasks. This view includes rich HTML descriptions at the workflow and task levels, expanding Jupyter's concept of the computational

narrative from a single Notebook to an entire scientific workflow. The Widget toolkit also provides task status, workflow control, stdout/stderr logs, file tracking and resource usage, providing quick access to relevant job information without leaving the Notebook.

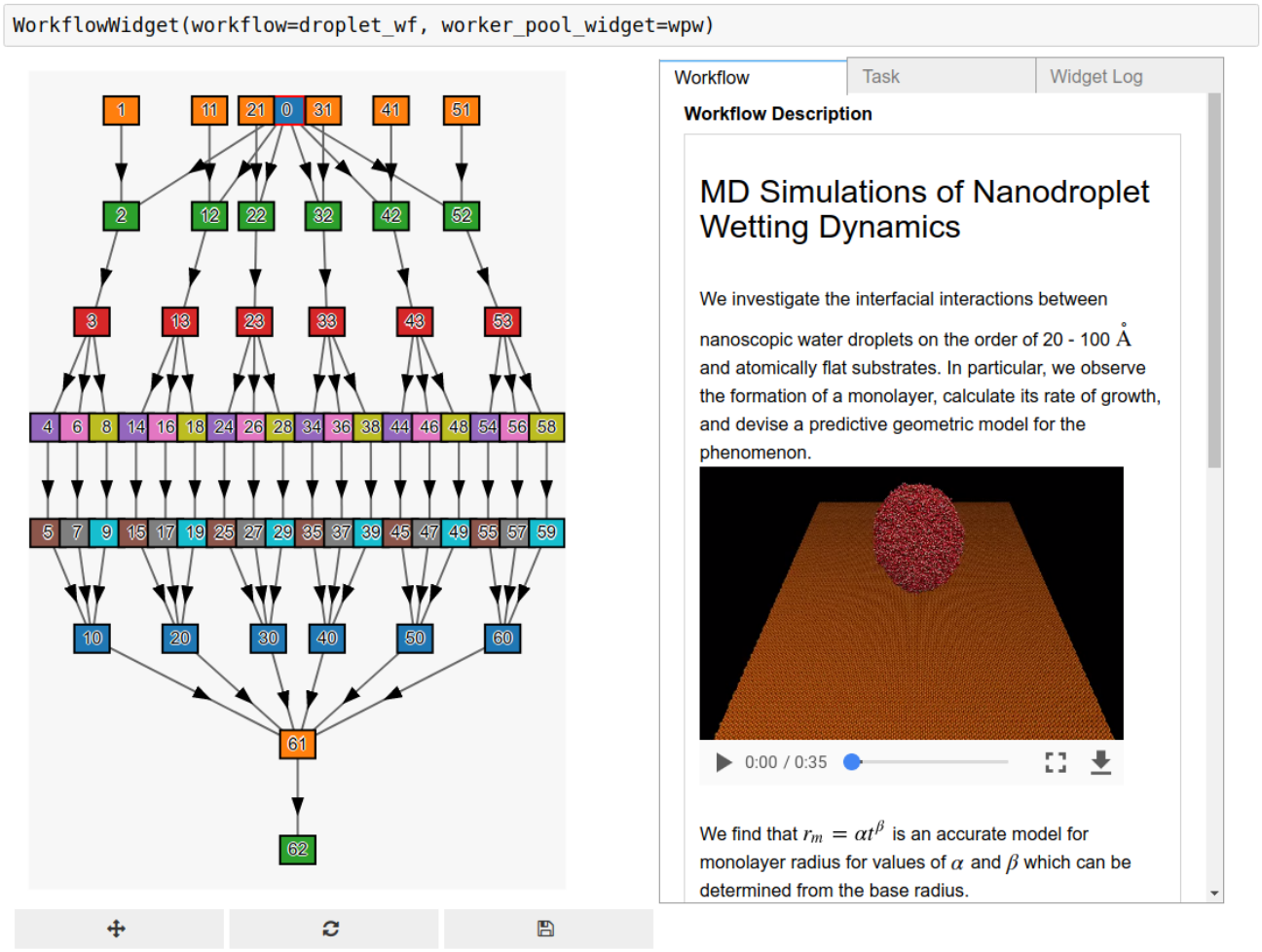

Fig. 2. LAMMPS Workflow with iPyWidgets interfacing with Kale.

V. Workflow Wrappers

In order to make Kale as easy to use as possible we introduce the idea of workflow wrappers. Kale allows you to define a workflow using its own object oriented Python syntax, but you can also use existing workflow tools (Parsl [7], Fireworks [4] etc.) or job execution frameworks (IPyParallel [5], Dask [6] etc.) to define your workflow. We are creating simple wrappers and interceptors that will allow you to take existing functions and tasks defined using these frameworks - by adding a little bit of extra code to wrap your workflow, you can now add a Kale layer to this workflow.

Kale remains agnostic to the underlying framework or the actual batch execution system. Once the task is launched, the Notebook communicates directly with the worker service through the REST API independent of the backend.

VI. Use Case

As part of our design philosophy we seek to ground our work in real-world use cases. As an example, we describe how we are currently using the Kale system to meet the needs of deep learning workflows for particle physics at NERSC [9]. In particular, Jupyter Notebooks in conjunction with Kale can be used for two different kinds of deep learning tasks.

Hyper-parameter optimization: This involves using a Jupyter Notebook to optimize a set of hyper-parameters (such as number of hidden layers in the neural network or learning rate). We configure the set of desired hyper-parameters that we wish to search over using a Widget interface, and then launch a series of model training runs on the HPC backend across a number of different nodes. The jobs themselves are wrapped as Kale tasks, and we can view a model output dashboard with current best and worst model runs in real-time directly in a Notebook Widget.

Monitoring specific training runs: For a given set of hyper-parameters, we can monitor and control model training runs, including live visualization of the loss function through Notebook Widgets communicating with Kale. Enabling human intervention allows for poorly performing runs to be halted and for new runs to be started in optimal regions of the parameter space, maximizing productive utilization of computational resources.

VII. Conclusion and Future Work

We believe that a system like Kale plays a key role in connecting interactive notebook environments like Jupyter with large scientific workflows at scale, and adds a key human-in-the-loop component to this process. This system combines the usability and convenience of a web science gateway for HPC, with the flexibility of a rich programming environment to enable ad-hoc exploratory data analysis. Moving forward, we expect to develop Kale in conjunction with multiple science use cases such as high-throughput materials discovery and automated analysis and classification of cosmology images. Better integration with different cloud and HPC environments is another potential area of investigation.

In conclusion, we hope that Jupyter along with Kale can help provide interactivity and iteration to what was previously a very fractured and fragmented process, thus reducing the overall time to scientific insight.

Acknowledgment

This work was supported by Lawrence Berkeley National Laboratory, through the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] "Jupyter Notebooks - a publishing format for reproducible computational workflows." Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing. ELPUB (2016). doi: 10.3233/978-1-61499-649-1-87

[2] "Demo: Extending Jupyter to Support Interactive High Performance Computing.", Matthew Henderson, Oliver Evans, Shreyas Cholia, Fernando Pérez,Gateways 2017, November 2017, https://doi.org/10.6084/m9.figshare.5501137.v1

[3] "Poster: Science at the Speed of Thought: Enhancing Jupyter to Enable Interactive Human-in- the-loop Supercomputing", Matthew Henderson, Oliver Evans, Shreyas Cholia, Fernando Pérez, Jupytercon 2017, August 2017.

[4] Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015) FireWorks: a dynamic workflow system designed for high-throughput applications. Concurrency Computat.: Pract. Exper., 27: 5037–5059. doi: 10.1002/cpe.3505.

[5] IPyParallel. Interactive Parallel Computing in Python https://ipyparallel.readthedocs.io/

[6] Dask Development Team (2016). Dask: Library for dynamic task scheduling, URL http://dask.pydata.org

[7] Babuji, Y., Brizius, A., Chard, K., Foster, I., Katz, D.S., Wilde, M., & Wozniak, J.. (2017, August 30). Introducing Parsl: A Python Parallel Scripting Library. Zenodo. http://doi.org/10.5281/zenodo.853491.

[8] R. Thomas, S. Canon, S. Cholia, L. Gerhardt, and E. Racah. Toward Interactive Supercomputing at NERSC with Jupyter. Cray User Group (CUG) Conference Proceedings, May 2017.

[9] "Deep Neural Networks for Physics Analysis on low-level whole-detector data at the LHC." Wahid Bhimji, Steven Andrew Farrell, Thorsten Kurth, Michela Paganini, Prabhat, Evan Racah. Nov 9, 2017. 6 pp. Conference: C17-08-21, e-Print: arXiv:1711.03573