LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-792241

Interactive In Situ Visualization and Analysis using Ascent and Jupyter

S. Ibrahim, T. Stitt, M. Larsen, C. Harrison

October 2, 2019

SC19
Denver, CO, United States
November 17, 2019 through November 22, 2019

Disclaimer

Interactive In Situ Visualization and Analysis using Ascent and Jupyter

Seif Ibrahim*
University of California, Santa Barbara
Lawrence Livermore National Laboratory

Thomas Stitt†
Lawrence Livermore National Laboratory

Matthew Larsen‡
Lawrence Livermore National Laboratory

Cyrus Harrison§
Lawrence Livermore National Laboratory

ABSTRACT

There is a gap in interactive in situ solutions for HPC simulations. While we have access to fully featured visualization UIs through tools like ParaView Catalyst and VisIt LibSim, we lack in situ infrastructure to access more general interactive environments like Jupyter. The Jupyter ecosystem of tools provides a rich paradigm for interactive data analysis and is well suited to help expand interactive use of in situ. The complexity of the Jupyter ecosystem and HPC center security requirements pose challenges to develop software infrastructure that enables direct use of Jupyter in simulation codes.

With this work, we describe a system that enables simulations instrumented with Ascent to connect to Jupyter. This system allows simulation users to interact with their data in situ using Jupyter Notebooks. The system combines Ascent's embedded Python filter infrastructure with a Client/Server Jupyter Bridge Kernel design that simplifies both deployment and security considerations on HPC systems. We describe the design of this system, demonstrate basic usage, and describe a prototype Ascent rendering UI built on top of this system.

ACM Reference Format:
Seif Ibrahim, Thomas Stitt, Matthew Larsen, and Cyrus Harrison. 2019. Interactive In Situ Visualization and Analysis using Ascent and Jupyter. In *Proceedings of ISAV Workshop 2019 (ISAV'19).* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

HPC simulation users are accustomed to visualizing simulation data post-hoc using interactive tools. The portion of simulation data available for post-hoc exploration is shrinking due to compute power starting to greatly outpace I/O rates in HPC architectures. In situ tools help reduce I/O when a user's actions can be prescribed apriori to simulation, but more broadly a lack of human-in-the-loop interactive access remains a barrier to adoption of in situ tools. Despite the relative cost of I/O increasing rapidly, users prefer post-hoc workflows because they offer interactivity and flexibility. Interactive use is critical for many visualization operations, such as adjusting camera angles. Interactive exploration is also often necessary to select useful actions to apply in non-interactive contexts.

Allowing users to easily connect to a running simulation and interactively explore their data can help mitigate the perception that in situ tools are only viable for actions prescribed apriori. The Jupyter ecosystem of tools provides a rich paradigm for interactive data analysis. The Jupyter Notebook [10] is an open-source web application that provides a read–eval–print loop (REPL) interface to a language kernel, such as a Python interpreter, and interactive display of results in the web browser. The Notebook paradigm also helps users easily share both analysis logic and results. Jupyter is flexible and extremely extensible. Creating a path to use Jupyter for in situ analysis of simulation data would allow us to leverage community investment in Jupyter instead of creating and teaching our own new paradigms to simulation users. Jupyter combines interactive access to the Python ecosystem and client-side web technologies. The Python ecosystem provides a rich set of analysis tools used ubiquitously in both the scientific computing and data science communities. Client-side web technologies allow us to rapidly develop and easily deploy interactive user interaces. This combination will lower barriers to develop both single purpose bespoke analysis scripts and more complex custom solutions for specific user communities. To unlock the potential of in situ use of Jupyter for HPC simulations, there are two key challenges that need to be addressed:

- How do we make it easy for simulation code developers to connect simulation data to Jupyter?
- What can we easily deploy in HPC centers?

In this work we extend Ascent to address these challenges and provide a way to connect to Juptyer Notebooks and explore simulation data in situ. Ascent [14] is a flyweight in situ infrastructure designed for leading-edge supercomputers that supports both distributed-memory and shared-memory parallelism. We selected Ascent for this work because of its flyweight design and its built-in distributed-memory Python environment. Paraview Catalyst [4] and VisIt LibSim [18] are in situ solutions built on top of existing full featured visualization tools. They support interactive use with

*email: ibrahim5@llnl.gov
†email: stitt4@llnl.gov
‡email: larsen30@llnl.gov
§email: cyrush@llnl.gov

ISAV'19, November 2019, Denver, CO, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

in situ data via the user interfaces of these full featured tools. The Paraview and VisIt teams invested substantial resources over a long period of time to develop these user interfaces. Ascent's design supports connecting to Catalyst and LibSim to leverage these user interfaces and Ascent currently provides a path to connect to Catalyst. While we will support connecting to these traditional full featured interfaces via Ascent, we see a large opportunity to expand interactive use by connecting simulation codes to Jupyter Notebooks.

Our primary goal for this work is to provide a simple path to connect a simulation code written in any language that Ascent's API supports (C++, C, Fortran, or Python) to Python-based Jupyter Notebooks. The goal is general interactive access to Python in Jupyter. Additionally, Ascent's API is designed for batch and scripting use cases, so a REPL-based Jupyter Notebook provides a natural mechanism for interactive access to Ascent as well.

In this paper, we outline two important contributions that enable Jupyter Notebooks with interactive in situ access to data:

- The Bridge Kernel Strategy, a simple secure solution for connecting to Jupyter
- A Bridge Kernel integration into Ascent, allowing any simulation code with an Ascent integration to leverage Jupyter

2 BACKGROUND

Ascent provides a built-in distributed-memory Python environment. To connect to Jupyter we leverage the Bridge Kernel strategy, outlined in Section 3.1, inside of Ascent's Python environment. This section provides background on the solution space we surveyed while developing this solution. Our target HPC simulations are all MPI-based, so we focus on solutions involving Jupyter and MPI.

2.1 MPI and Jupyter

Ascent's distributed-memory Python environment leverages mpi4py [5–7]. We selected mpi4py because it is widely adopted and is a feature complete Python MPI package.

Looking at MPI solutions in the Jupyter space, IPyParallel [1] is a widely used interactive distributed computing framework for Jupyter. It provides a cluster abstraction and a specific cluster instance that supports MPI. A MPI-enabled Python-based simulation code can use IPyParallel directly and leverage Ascent's Python API for interactive in situ visualization, however this approach will not work for C++, C, or Fortran-based simulation codes. Another barrier for using IPyParallel with a simulation code is that the primary MPI job needs to be an instance of the IPython Parallel cluster engine. For ease of integration and use, we chose to develop a solution that does not change how simulation code is launched.

Dask [8, 16] is a widely used and flexible distributed computing framework for Python. Dask.distributed scales Dask to parallel clusters and dask-mpi provides a way to setup Dask workers using MPI. Dask.distributed is also integrated with IPyParallel, allowing Dask workers to use an IPyParallel cluster. For interactive use, Dask provides a client that dispatches Dask commands to a Dask cluster simplifying using Dask in Jupyter Notebooks. This client is specific to the Dask API and does not provide general command forwarding. While the Dask client does not provide the general MPI support we are aiming for, in future work we are very interested in learning how to use Dask for analysis within our distributed-memory Python environment.

2.2 Deployment in HPC Centers

JupyterHub [2] is the primary solution for multi-user hosting of Jupyter Notebooks and it shows great promise as a solution to support general interactive HPC [15]. It is in use at several HPC Centers, including NERSC, LLNL's LC, the OLCF and the ALCF.

JupyterHub provides user authentication and dynamically spawns Jupyter Notebook servers and kernels. The communication protocol between Jupyter Notebook servers and kernels uses up to 5 socket connections [3] and additional sockets are used for communication between the Notebook server and the user's browser. On HPC clusters, these sockets need to be secure and authenticated to preserve user permissions and protect access. At LLNL, we invested to extend JupyterHub to secure these connections by modifying JupyterHub so that a certificate authority signs and issues certificates on-the-fly to encrypt all channels from the Notebook server to the user's browser. In addition, because messaging between the Notebook server and the kernel is unencrypted we configure JupyterHub to enforce IPC (inner-process communication) for kernel messaging to protect client-kernel communication with Unix's standard filesystem permissions. More details were presented in the poster [17]. These changes were contributed back to JupyterHub.

3 SYSTEM DESIGN

This section outlines the details of the Bridge Kernel strategy and how we leverage it inside of Ascent to provide an in situ connection to Jupyter.

3.1 Bridge Kernel

The Python Bridge Kernel allows you to connect any Python-enabled code to Jupyter. In this paper we focus on using the Bridge Kernel to connect Ascent to Jupyter, however this same approach can be used with any application or library that provides an embedded Python interpreter (e.g., a simulation code, VisIt's command line interface, etc). The Bridge Kernel strategy was first developed and implemented at LLNL in C++ to connect an embedded Lua interface from a C++ and FORTRAN simulation code to a Jupyter Notebook. Given the broad interest in interactive solutions for Python, we created a Python variant of the Bridge Kernel. This variant was released open source and is available at [13].

The Bridge Kernel uses a Client/Server design. The Client portion is a proper Jupyter kernel, built using Jupyter's Python kernel infrastructure modules. The Server portion only depends on a basic Python install and mpi4py, it does not directly use Jupyter's kernel infrastructure. The Bridge Kernel Client and Server communicate over a single secure socket connection. The socket is secured using either an IPC (named Unix/file-system) socket, or a TLS-encryped IP socket. The IPC option writes a Unix socket to the user's home directory that is protected with standard unix filesystem permissions. The TLS option creates a certificate authority (CA) that generates a set of encryption keys to secure messages and authenticate connections. The client gets the keys and the CA's public key, uses the CA to verify the server, and uses the key pair to encrypt messages. Upon connection, the server verifies the client's public key

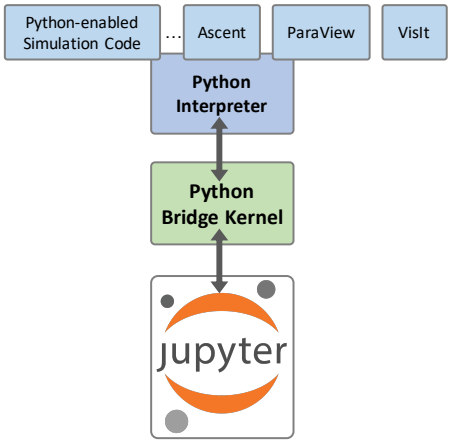

Figure 1: Any application or library that provides a Python interpreter can use a Python Bridge Kernel to connect to Jupyter.

through its CA. In LLNL's JupyterHub installation Notebook servers are launched on login nodes while applications generally run on compute nodes. To support this common case, the Bridge Kernel automatically creates ssh tunnels for either the IP or IPC socket.

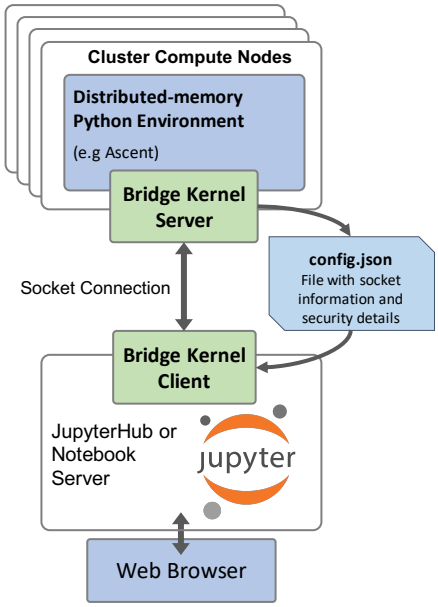

Figure 2: The Client/Server Bridge Kernel design provides the secure connection necessary for deployment on HPC systems. It supports a UDP/IPC mode using native unix permission controls or an OpenSSL mode for encrypted communication over a regular socket.

Because the server exists as a minimal Python module, a simulation code and Ascent can be built against a Python without the Jupyter stack. This flexibility is important for deployment because HPC centers may select a different version of Python to deploy JupyterHub than a simulation code is compatible with. It also gives the simulation code developers the freedom to manage the Python modules installed for their analysis, instead of relying on a system install.

3.2 Ascent Integration

Ascent provides *Extracts* that allow users to capture data for use outside of Ascent. The Jupyter connection is provided by a new Jupyter Extract that is a simple extension of Ascent's existing Python Extract. Ascent's Python Extract executes a Python script in an embedded Python interpreter that has zero-copy access to data from Ascent. The data passed into the Python interpreter is described using Conduit [12] and is accessed using Conduit's Python API. This provides a path for a simulation code written in any of Ascent's supported language APIs (C++, C, Fortran, or Python) to publish data and manipulate it using Python. When using MPI, Ascent provides a distributed-memory Python environment that can leverage mpi4py for coordination between MPI tasks. As shown in Figures 2 and 3, Python code sent through Jupyter executes on all MPI tasks, where each task has a subset of the data published by the simulation.

The Jupyter Extract runs the server portion of the Bridge Kernel in Ascent's Python environment, waiting for a connection from the Bridge Kernel client. After the client connects, it forwards commands from the Jupyter Notebook for execution in Ascent's Python environment. When the client disconnects control returns back to Ascent and then to the Simulation.

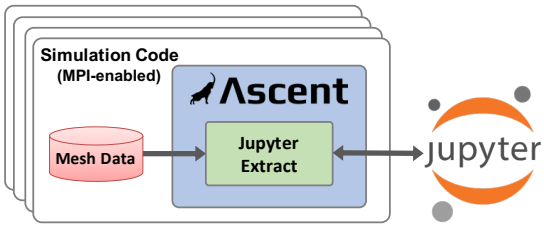

Figure 3: Ascent's Jupyter Extract gives users an interactive way to analyze and visualize their simulation data in situ using Jupyter Notebooks.

4 EXAMPLES

In this section we share two interactive uses of Ascent's Jupyter Extract.

4.1 Running an Ascent Python Extract Example in a Jupyter Notebook

Our first example demonstrates interactive Python coding and plotting in a Notebook. We compute and display a distributed-memory histogram of an energy field in the Cloverleaf proxy simulation. This is an extension of Ascent's Custom Python Extract Tutorial Example [11].

First, to use the Jupyter Extract we create an actions Node that requests an extract of type `jupyter`. Actions can be specified directly where Ascent is called in the simulation code, or

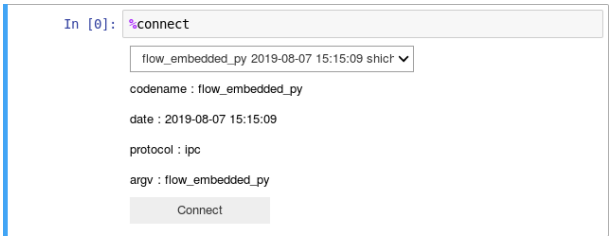

Figure 4: The Jupyter widget for connecting to a simulation.

they can be selected at runtime using a `ascent_actions.yaml` or `ascent_actions.json` file. In our case we use an `ascent_actions.yaml` file, with contents as shown in Listing 1.

```
-
  action: "add_extracts"
  extracts:
    e1:
      type: "jupyter"
```

Listing 1: Example ascent_actions.yaml actions file used to execute a Jupyter Extract.

We place the `ascent_actions.yaml` in the same directory as the Cloverleaf executable and launch Cloverleaf using MPI. After Cloverleaf is running, we launch the Jupyter Notebook server and start an instance of the Ascent Bridge Kernel. We execute the ***%connect*** magic, which brings up a simple UI to select and connect to the running Ascent Bridge Kernel server. Figure 4 shows a screenshot of the connection UI.

After connecting successfully, code typed in the Notebook is forwarded for execution on the Bridge Kernel server and any output is returned to the Notebook. Finally, we type and execute the Custom Python Extract script source and add code to plot the results using Matplotlib [9]. Figure 5 shows a screenshot of the Notebook used to run this example.

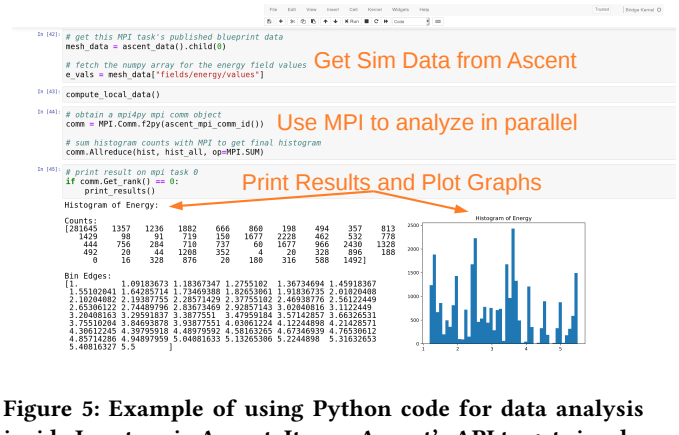

Figure 5: Example of using Python code for data analysis inside Jupyter via Ascent. It uses Ascent's API to get simulation data, constructs a histogram in parallel using mpi4py, and finally plots the results with Matplotlib.

4.2 Prototype Rendering UI for Ascent

To demonstrate the possibilities for bespoke or customized interfaces, our second example leverages more aspects of the web browser to create an interactive UI for Ascent rendering.

We build a prototype UI using custom Jupyter UI Widgets and a Client/Server model to communicate with Ascent. We extend the Ascent Bridge Kernel to create a protocol for passing messages specifically between Jupyter and Ascent. We use JSON to synchronize details about pipelines, scenes, camera parameters, etc., between the front-end client and Ascent. This design separates the analysis code running in Ascent from the front-end code running in Jupyter and preserves our ability to leverage separate software stacks for the client and server.

Our custom Jupyter Widget uses a Model-view-controller (MVC) design and is written using a combination of Python (model) and HTML/CSS/Javascript (view, controller). The widget runs WebGL in the browser, rendering a draggable cube which provides trackball controls for changing the camera postion. The camera postion of the WebGL cube is sent to Ascent which raytraces a corresponding image and sends the result to Jupyter where the user can view it and iterate to make further changes. The UI also responds to keypresses and button clicks so that users can fly around their simulation data in 3D. Figure 6 shows a screenshot of our prototype.

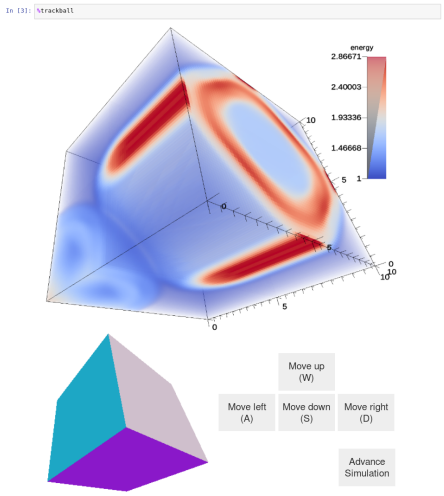

Figure 6: Trackball controls in Jupyter provide an interactive WebGL cube to modify the camera position for a volume plot rendered using Ascent in a Cloverleaf simulation.

5 CONCLUSION

In this paper we presented a system that extends Ascent to enable simulation users to run interactive Jupyter Notebooks with in situ access to their simulation data. Our key contributions are the Bridge Kernel design, which provides a general solution that simplifies interfacing with Jupyter and the use of the Bridge Kernel in Ascent to support a new Jupyter Extract. Further, we demonstrated two interactive uses of this new capability.

6 ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC (LLNL-CONF-ZZZZZZZ).

Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow other to do so, for United States Government purposes.

REFERENCES

[1] 2019. *IPython Parallel Documentation.* https://ipyparallel.readthedocs.io/
[2] 2019. *JupyterHub Documentation.* https://jupyterhub.readthedocs.io/
[3] 2019. *Messaging in Jupyter.* https://jupyter-client.readthedocs.io/en/stable/messaging.html
[4] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization.* ACM, 25–29.
[5] L. Dalcin, P. Kler, R. Paz, , and A. Cosimo. 2011. Parallel Distributed Computing using Python. In *Advances in Water Resources.* 34(9):1124–1139. https://doi.org/10.1016/j.advwatres.2011.04.013
[6] L. Dalcin, R. Paz, and M. Storti. 2005. MPI for Python. *J. Parallel and Distrib. Comput.*, 65(9):1108–1115. https://doi.org/10.1016/j.jpdc.2005.03.010
[7] L. Dalcin, R. Paz, M. Storti, and J. D'Elia. 2008. MPI for Python: performance improvements and MPI-2 extensions. *J. Parallel and Distrib. Comput.*, 68(5):655–662. https://doi.org/10.1016/j.jpdc.2007.09.005
[8] Dask Development Team. 2016. *Dask: Library for dynamic task scheduling.* https://dask.org
[9] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55
[10] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks - A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87
[11] Lawrence Livermore National Laboratory. 2019. *Ascent Documentation - Custom Python Extract Example.* https://ascent.readthedocs.io/en/latest/Tutorial.html#demo-4-custom-python-extracts
[12] Lawrence Livermore National Laboratory. 2019. *Conduit: Simplified Data Exchange for HPC Simulations.* https://software.llnl.gov/conduit/
[13] Lawrence Livermore National Laboratory. 2019. *LLNL Python Bridge Kernel Repository.* https://github.com/LLNL/bridge-kernel
[14] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. 2017. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17).* ACM, New York, NY, USA, 42–46. https://doi.org/10.1145/3144769.3144778
[15] Michael Milligan. 2018. Jupyter as Common Technology Platform for Interactive HPC Services. *CoRR* abs/1807.09929 (2018). arXiv:1807.09929 http://arxiv.org/abs/1807.09929
[16] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.
[17] Thomas Stitt, Thomas Mendoza, Cyrus Harrison, and Todd Gamblin. 2018. Addressing Barriers to In-situ Use of Jupyter Notebooks in HPC Simulations.
[18] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel in Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV).* 101–109.