BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

Jupyter + Kale: Human-in-the-loop Interactivity in HPC Workflows

Shreyas Cholia, Matthew Henderson, Oliver Evans, Fernando Pérez

Lawrence Berkeley National Laboratory,

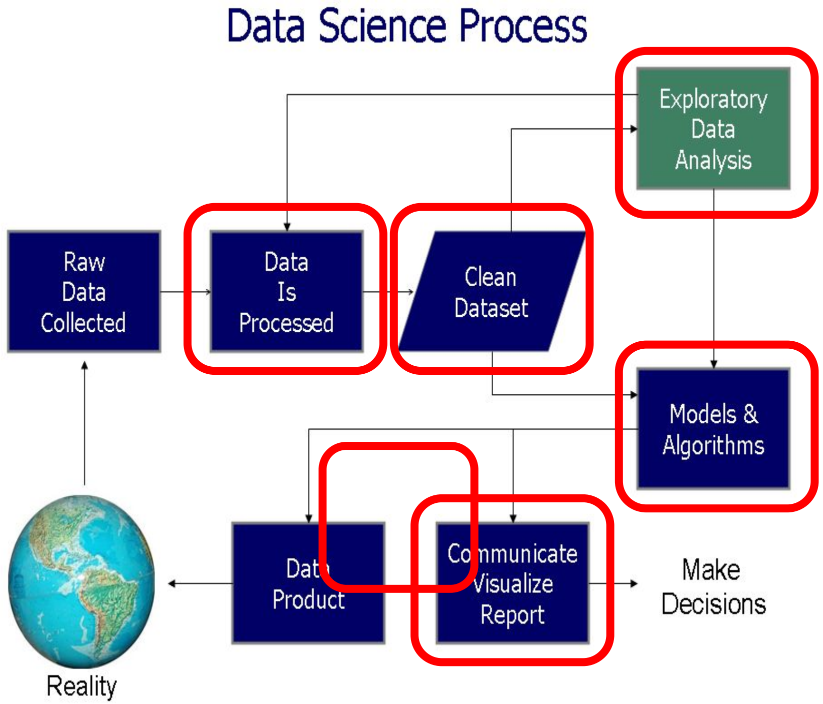
Gateways 2018 - Wednesday, September 26

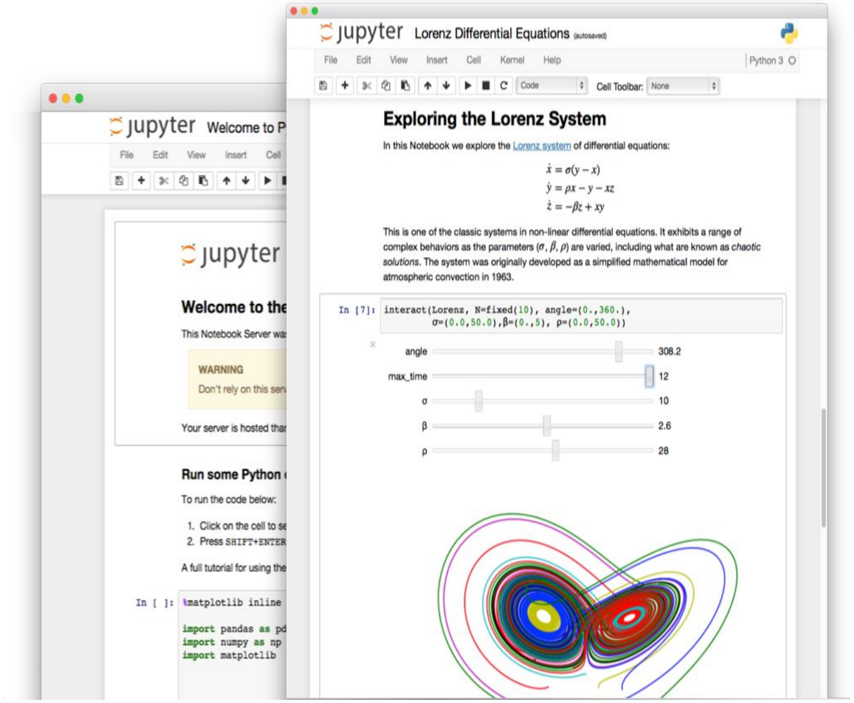
What is Jupyter?

Tool for reproducible, shareable narratives, literate computing:

***Notebook*: Document containing code, comments, outputs.**

Rich text, interactive plots, equations, widgets, etc.





Jupyter Welcome to P

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press `SHIFT+ENTER`

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Jupyter Lorenz Differential Equations (autosaved)

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

angle 308.2

max_time 12

σ 10

β 2.6

ρ 28

Why Now?

Data 8: Foundations of Data Science

Integral part of Big (Data) Science & Superfacility:

LSST-DESC, DESI, ALS, LCLS, Materials Project…
Kale LDRD (workflows), KBase...

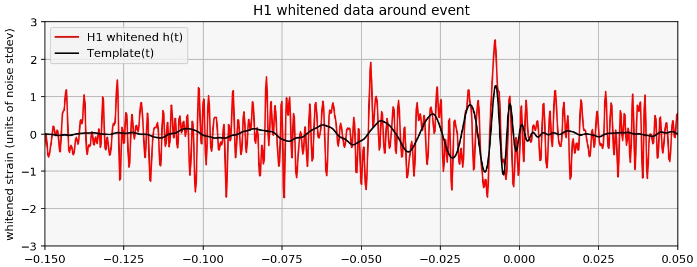
Generational shift in analytics for science + more:

UCB's Data Science 8 course, entirely in Jupyter
"I'll send you a copy of my notebook"
Training events adopting notebooks (DL)

Supporting reproducibility and science outreach:

Open source code and open source science
Jupyter notebooks alongside publications (LIGO)

2017 ACM Software System Award: "… ***a de facto standard for data analysis in research, education, journalism and industry.*** Jupyter has broad impact across domains and use cases. Today more than ***2,000,000 Jupyter notebooks are on GitHub***, each a distinct instance of a Jupyter application—covering a range of uses from technical documentation to course materials, books and academic publications."



LIGO Binary BH-BH Merger GW Signature
Figure from LIGO EPO/Publication Jupyter Notebook

Jupyter Gateway Deployments

Many science gateway environments now support Jupyter Notebooks

- Enable custom, ad-hoc analyses on scientific data
- **Jupyterhub** lets you deploy multi-user notebook environments
- **Jupyterlab** enables integration across "apps"
- Deployments at NERSC, OSU, BNL, XFEL, TACC, Pacific Research Platform etc.

Motivation

Improved scientific discovery and productivity through better tools

- Enable **human-in-the-loop** computing
- Enhance reproducibility and collaboration

Enable exploratory data analytics, deep learning, workflows, and more through Jupyter on NERSC and other HPC systems.

How are scientists using Jupyter in HPC?

QA/QC

- Generate notebooks from HPC output
- Human inspection
- Iterate on steps

Master Workflow Controller

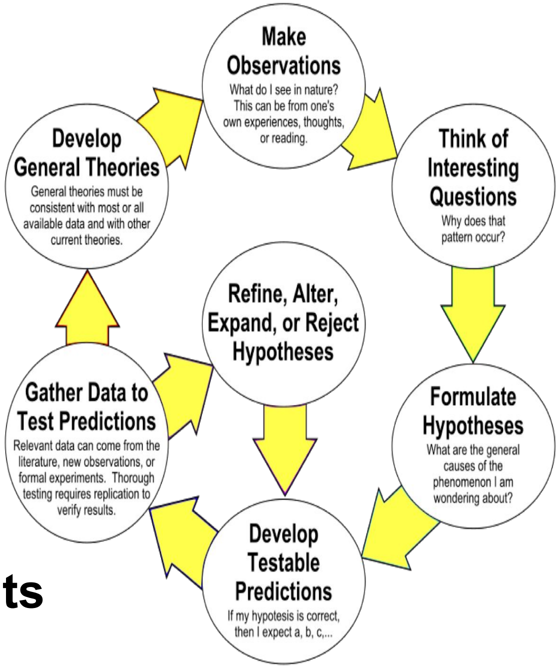
- Setup and control job workflows through notebook.
- Use batch queue to run jobs and use notebook before & after job steps

Parallel/Distributed interactive work

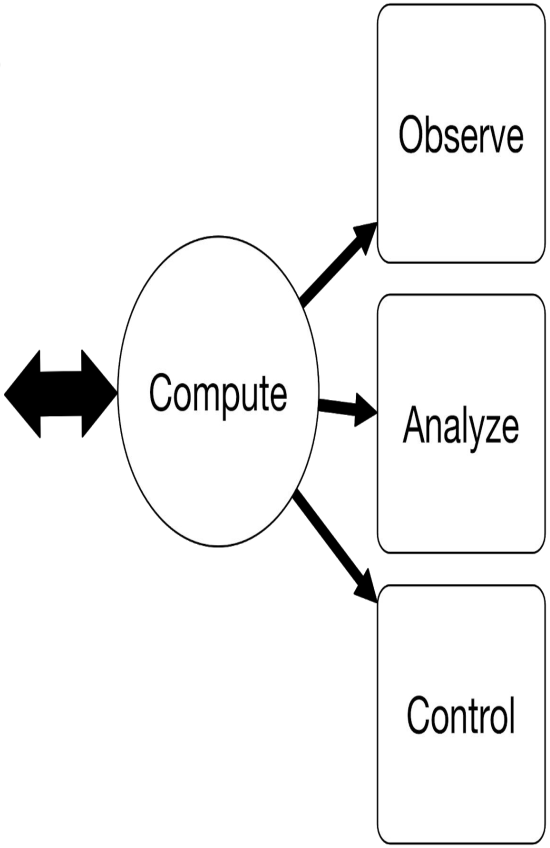
- Scaling up single-node notebook operations to a parallel/distributed mode
- Request HPC nodes
- Jupyter on Master Node + Workers
  - e.g., IPyparallel, Dask
- Live control using Notebooks

Our Focus

The Scientific Method as an Ongoing Process


Make Observations
What do I see in nature? This can be from one's own experiences, thoughts, or reading.
Think of Interesting Questions
Why does that pattern occur?
Formulate Hypotheses
What are the general causes of the phenomenon I am wondering about?
Develop Testable Predictions
If my hypotesis is correct, then I expect a, b, c,...
Gather Data to Test Predictions
Relevant data can come from the literature, new observations, or formal experiments. Thorough testing requires replication to verify results.
Develop General Theories
General theories must be consistent with most or all available data and with other current theories.
Refine, Alter, Expand, or Reject Hypotheses

https://en.wikipedia.org/wiki/Scientific_method



Provide a more natural development cycle for scientists using HPC

Human-in-the-loop

- Real-time task monitoring
- Dynamic task control
- Runtime ad-hoc analyses
- Seamless cycle between code results and viz

Our approach

Leverage Jupyter architecture

- Notebooks
- Widgets
- Kernels
- Distributed Execution

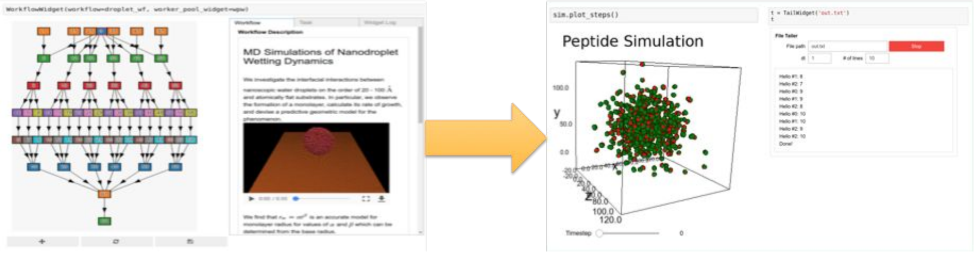
Extend the Jupyter ecosystem

- Fine-grain Task Control
- Task Monitoring
- Real-time interaction

Kale: Human-in-the-loop HPC

Project Kale is a research effort focused on adapting the Jupyter machinery for HPC workflows

View, Control, Monitor

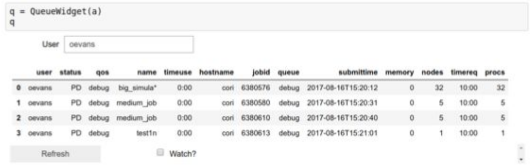


```
WorkflowWidget(workflow=droplet_wf, worker_pool_widget=wpw)
```

MD Simulations of Nanodroplet Wetting Dynamics

```
sim.plot_steps()
```

```
t = TailWidget('out.txt')
t
```



```
q = QueueWidget(a)
q
```

User oevans

	user	status	qos	name	timeuse	hostname	jobid	queue	submittime	memory	nodes	timereq	procs
0	oevans	PD	debug	big_simula*	0:00	cori	6380576	debug	2017-08-16T15:20:12	0	32	10:00	32
1	oevans	PD	debug	medium_job	0:00	cori	6380580	debug	2017-08-16T15:20:31	0	5	10:00	5
2	oevans	PD	debug	medium_job	0:00	cori	6380610	debug	2017-08-16T15:20:40	0	5	10:00	5
3	oevans	PD	debug	test1n	0:00	cori	6380613	debug	2017-08-16T15:21:01	0	1	10:00	1

- Master notebook to control workflow
- Jupyter notebooks as **interactive workflow steps**
- Interaction with workflow tasks via kernels
- Realtime Monitoring of HPC jobs and output
- Widgets and dashboards for batch job management

Control and Monitor Tasks

HPC tasks are wrapped by a process

- Non-invasive to the task

The process provides (via REST API)

- Resource monitoring
  - Task level + Node level
- Task control
  - Start, Stop, Pause, Resume
- Extend to wrap tasks with arbitrary callouts

Not Another Workflow System

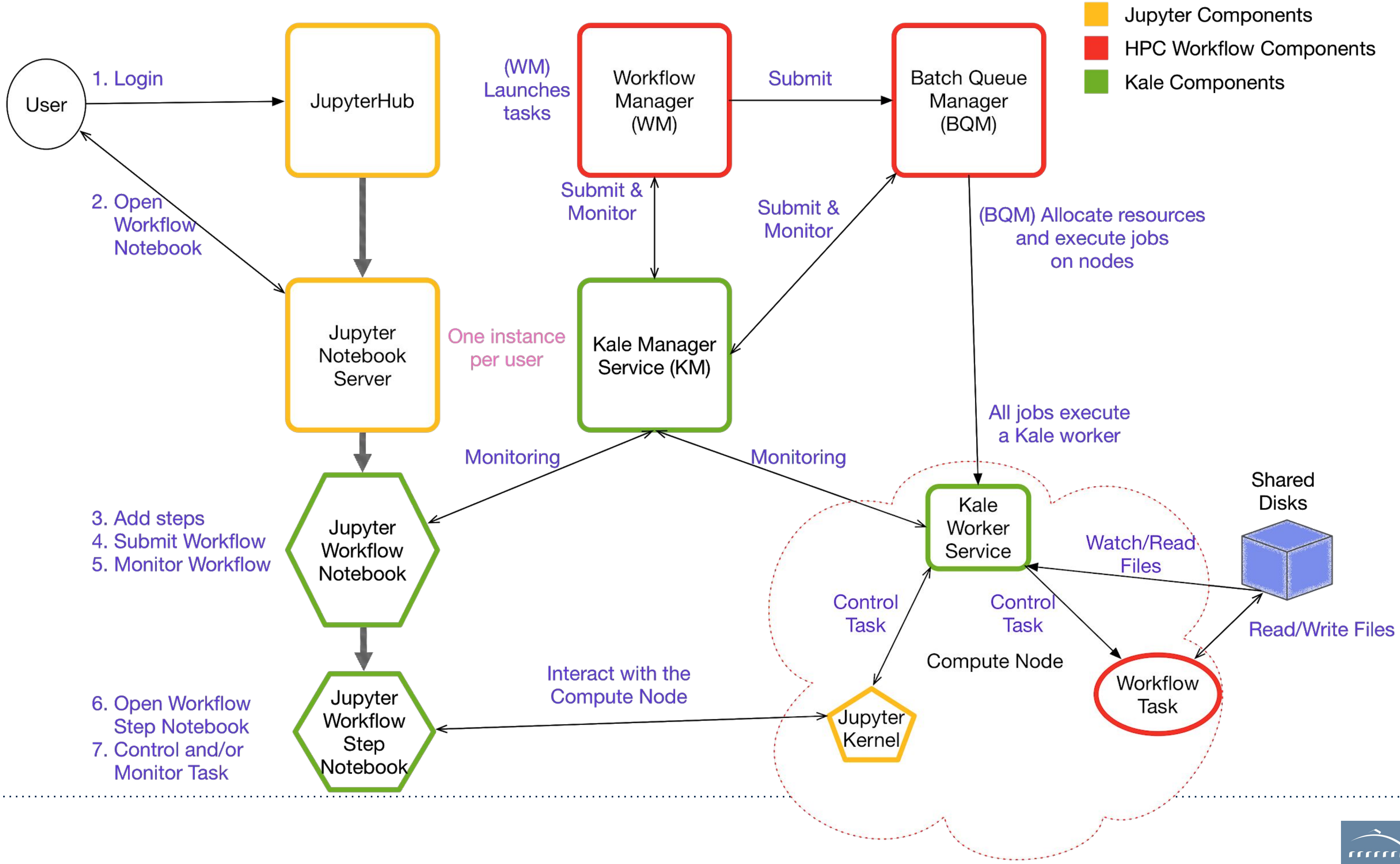
Wikipedia page on workflow systems: 100+ packages

We don’t need another workflow manager.

Instead Kale hooks into existing workflow or task execution systems

- Fireworks, IPyParallel, Parsl etc.

Overall system



🟨 Jupyter Components

🟥 HPC Workflow Components

🟩 Kale Components

User

1. Login

JupyterHub

(WM) Launches tasks

Workflow Manager (WM)

Submit

Batch Queue Manager (BQM)

2. Open Workflow Notebook

Submit & Monitor

Submit & Monitor

(BQM) Allocate resources and execute jobs on nodes

Jupyter Notebook Server

One instance per user

Kale Manager Service (KM)

All jobs execute a Kale worker

Monitoring

Monitoring

3. Add steps
4. Submit Workflow
5. Monitor Workflow

Jupyter Workflow Notebook

Kale Worker Service

Shared Disks

Watch/Read Files

Control Task

Control Task

Read/Write Files

Compute Node

6. Open Workflow Step Notebook
7. Control and/or Monitor Task

Jupyter Workflow Step Notebook

Interact with the Compute Node

Jupyter Kernel

Workflow Task

A Word About Python

- Jupyter has a close connection with Python (emerged from IPython)
- And many of the tools in the Jupyter Ecosystem are centered around Python
- Scientists seem to really like it to drive their workflows, so we focus a lot of development here
- Kale can be used to wrap any arbitrary process so we aren’t limited to Python codes (but our examples will focus on a Python backend)

Use Case: Deep Learning on HPC

- Configure a set of hyperparameters
- Launch HPC model training runs
- View a model output dashboard with current best and worst model runs
- Manage Distributed Training

Control model runs
- Stop poor performers
- Start new models exploring different parameter spaces

Our approach

Wrap execution of model runs

Build the UI with Jupyter Widgets for use in a Notebook

Features

- Configure hyperparameters
- Submit HPC runs
- Display current Best/Worst models
- Controls for model execution

Ecosystem

Task distribution and management

IPython Parallel (ipyparallel) - Hub and Controller communicate with a set of **ipyparallel engines** (ipython kernels running across multiple nodes). Publish data that is monitored via background threads and event listener.

Currently single controller bottleneck but only for notebook communication - can use other MPI libraries like Horovod for bulk communication alongside

See also: **Dask, Horovod**

Live Plotting, Interactive visualization, Realtime Communication

IPyWidgets - Real-time interaction with Jupyter backend, live rendering of data, Start/Stop Tasks

QGrid (Quantopian) - Interactive tables with sort, filter, row selection; Updated in real time

BQPlot (Bloomberg) - Live Plotting and interaction with QGrid

Fine grained hooks into resources

Kale - Extends jupyter ecosystem with manager and worker service that wrap backend task to provide fine-grain task control and node resource monitoring

Jupyter architecture

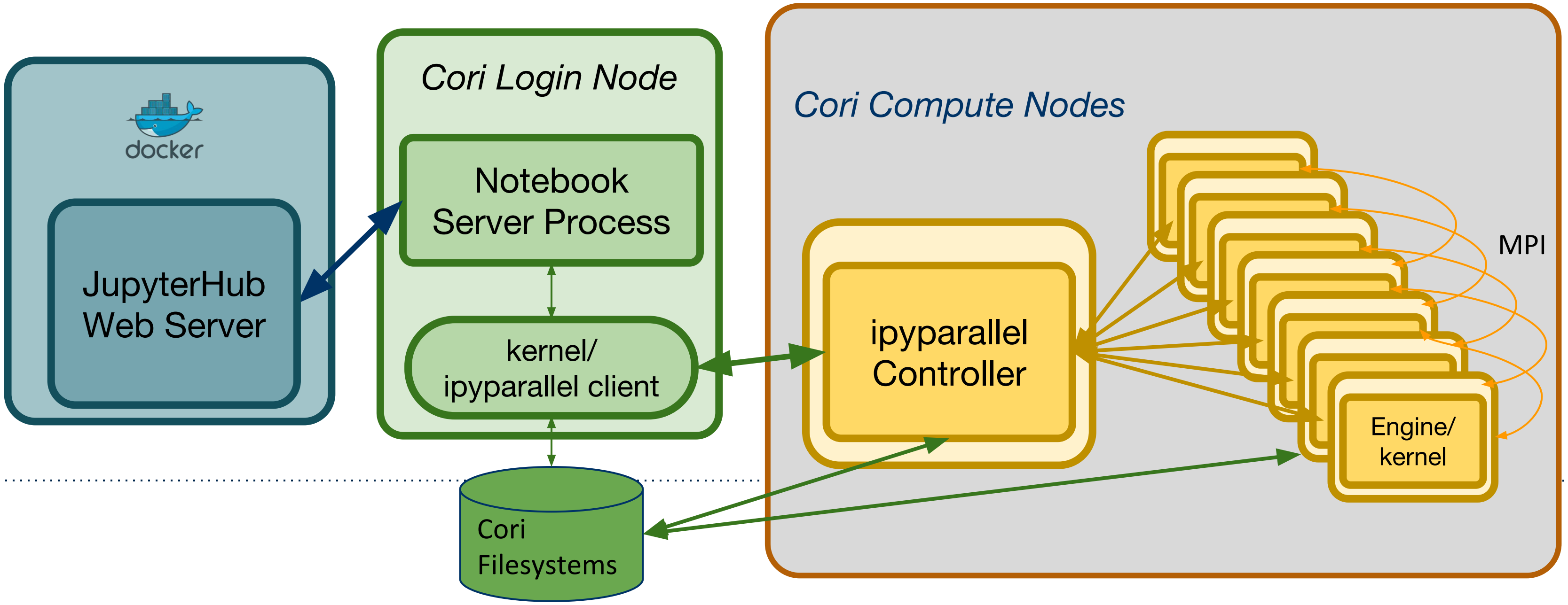
Allocate nodes on Cori interactive queue and start ipyparallel or Dask cluster

- Developed %ipcluster magic to setup within notebook

Compute nodes traditionally do not have external address

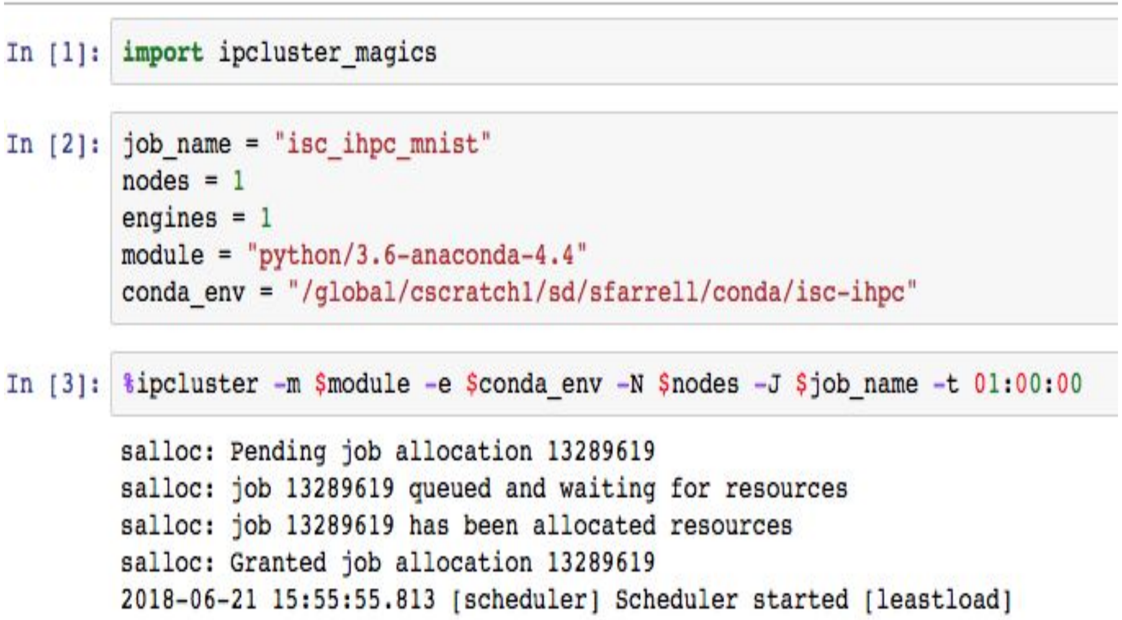
- Required network configuration / policy decisions

Distributed training communication is via MPI Horovod or Cray ML Plugin



Setting up ipyparallel cluster

Via Magic (entire workflow in notebook) or a console script

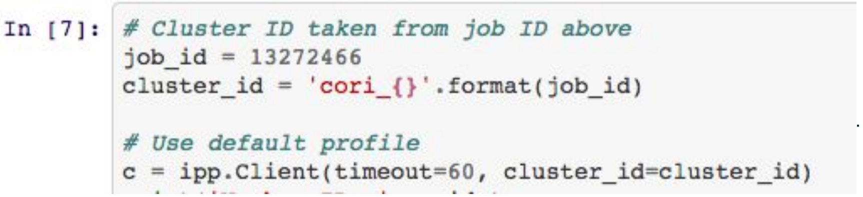


```
In [1]: import ipcluster_magics

In [2]: job_name = "isc_ihpc_mnist"
        nodes = 1
        engines = 1
        module = "python/3.6-anaconda-4.4"
        conda_env = "/global/cscratch1/sd/sfarrell/conda/isc-ihpc"

In [3]: %ipcluster -m $module -e $conda_env -N $nodes -J $job_name -t 01:00:00

        salloc: Pending job allocation 13289619
        salloc: job 13289619 queued and waiting for resources
        salloc: job 13289619 has been allocated resources
        salloc: Granted job allocation 13289619
        2018-06-21 15:55:55.813 [scheduler] Scheduler started [leastload]
```

salloc --qos=interactive -N 1 -C haswell
wbhimji@nid00032:~> ./startCluster.sh

```
# Use a unique cluster ID for this job
clusterID=cori_${SLURM_JOB_ID}
echo "Launching controller"
ipcontroller --ip="$headIP" \
     --cluster-id=$clusterID &
sleep 20
echo "Launching engines"
srun ipengine --cluster-id=$clusterID
```

Connect to cluster in notebook



```
In [7]: # Cluster ID taken from job ID above
        job_id = 13272466
        cluster_id = 'cori_{}'.format(job_id)

        # Use default profile
        c = ipp.Client(timeout=60, cluster_id=cluster_id)
```

Distributed Training

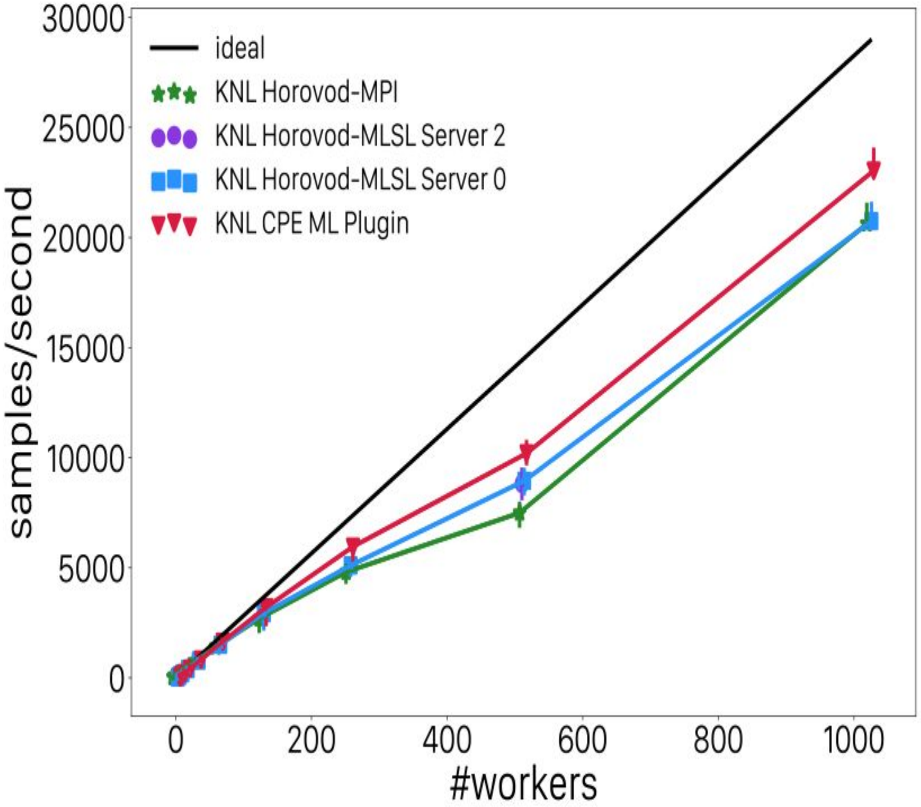
Kurth et al CUG 2018

Speed up training by parallelizing across nodes, e.g. for distributed *Stochastic Gradient Descent* (SGD) algorithms:

- Each node computes gradients locally
- Summed across nodes and propagated to all nodes (sync) or via parameter server (async)

MPI-based tools for distributed SGD now available :

- e.g Horovod and Cray PE ML Plugin



These methods/tools can scale well to many nodes on Cori (above is for a large image version of the same LHC CNN used here)

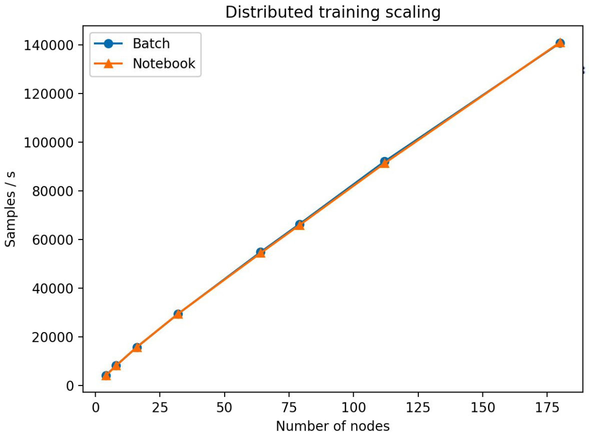
Distributed Training

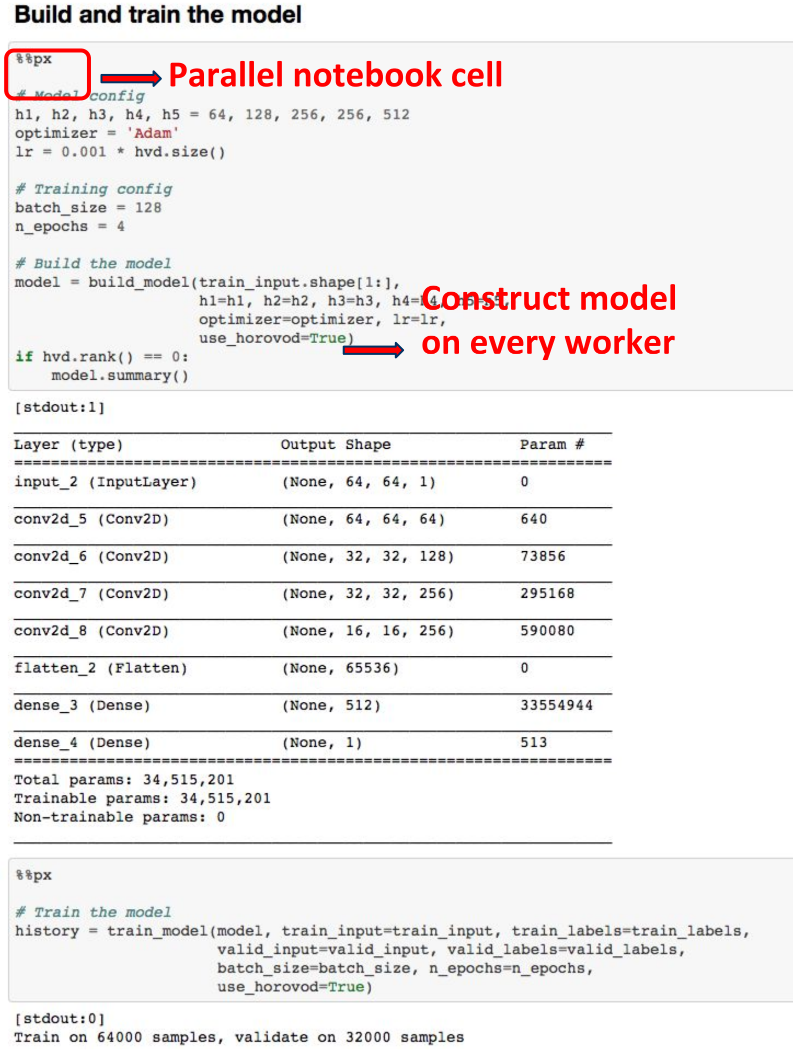
Distributed training in notebooks with IPyParallel and Horovod-MPI

Notebook cells specified for parallel execution using cell magic

- MPI code in a notebook

Scales well with no noticeable overhead from the notebook infrastructure





Build and train the model

```
In [8]: %%px

        # Model config
        h1, h2, h3, h4, h5 = 64, 128, 256, 256, 512
        optimizer = 'Adam'
        lr = 0.001 * hvd.size()

        # Training config
        batch_size = 128
        n_epochs = 4

        # Build the model
        model = build_model(train_input.shape[1:],
                            h1=h1, h2=h2, h3=h3, h4=h4, h5=h5,
                            optimizer=optimizer, lr=lr,
                            use_horovod=True)

        if hvd.rank() == 0:
            model.summary()
```

```
[stdout:1]
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64, 64, 1)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 64)        640
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 256)       295168
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 256)       590080
_________________________________________________________________
flatten_2 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               33554944
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 513
=================================================================
Total params: 34,515,201
Trainable params: 34,515,201
Non-trainable params: 0
_________________________________________________________________
```

```
%%px

# Train the model
history = train_model(model, train_input=train_input, train_labels=train_labels,
                      valid_input=valid_input, valid_labels=valid_labels,
                      batch_size=batch_size, n_epochs=n_epochs,
                      use_horovod=True)
```

```
[stdout:0]
Train on 64000 samples, validate on 32000 samples
```

Parallel notebook cell

Construct model on every worker

Train with Horovod on all workers

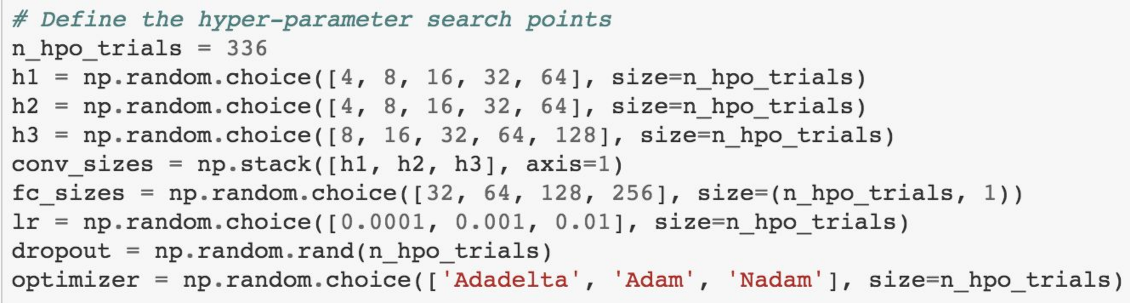
Distributed HPO

- Hyper-parameter optimization (HPO) algorithms are used to find a best set of possible model hyper-parameters
  - Can train and evaluate many models in parallel across nodes in HPC system
- Random Search HPO
  - Evaluate model at HP sets randomly sampled from a specified HP space
  - Simple algorithm; trivially parallelizable

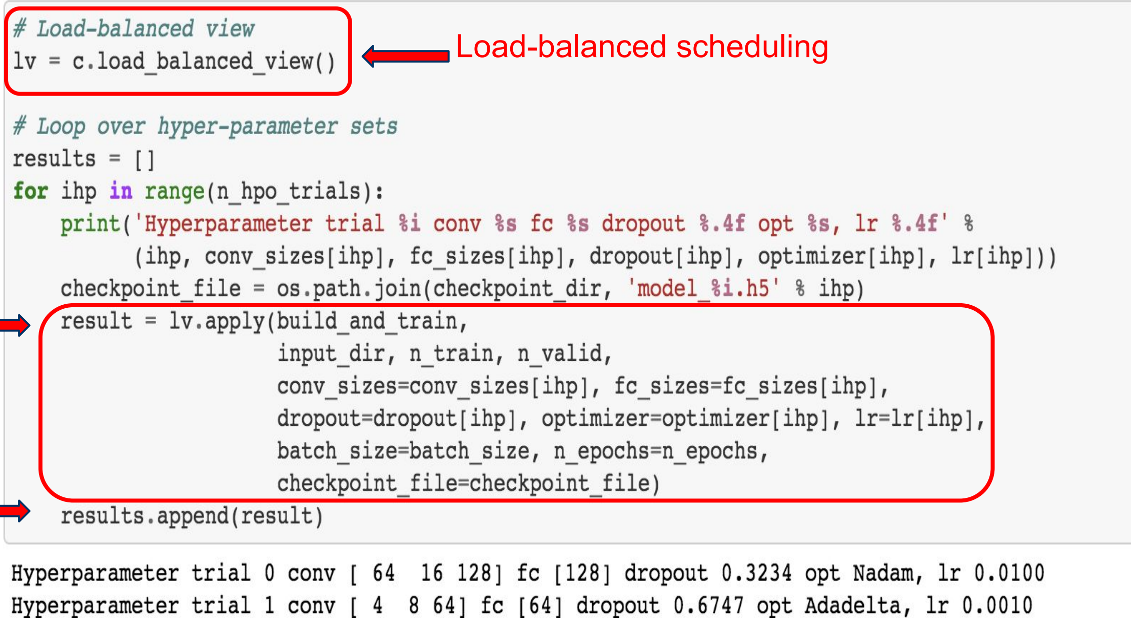
Distributed HPO - Setup

Easy but powerful setup for random search HPO

- Define HP sets to evaluate
- Define model training function
- Run the HPO tasks with load-balanced scheduler



```
# Define the hyper-parameter search points
n_hpo_trials = 336
h1 = np.random.choice([4, 8, 16, 32, 64], size=n_hpo_trials)
h2 = np.random.choice([4, 8, 16, 32, 64], size=n_hpo_trials)
h3 = np.random.choice([8, 16, 32, 64, 128], size=n_hpo_trials)
conv_sizes = np.stack([h1, h2, h3], axis=1)
fc_sizes = np.random.choice([32, 64, 128, 256], size=(n_hpo_trials, 1))
lr = np.random.choice([0.0001, 0.001, 0.01], size=n_hpo_trials)
dropout = np.random.rand(n_hpo_trials)
optimizer = np.random.choice(['Adadelta', 'Adam', 'Nadam'], size=n_hpo_trials)
```



```
# Load-balanced view
lv = c.load_balanced_view()

# Loop over hyper-parameter sets
results = []
for ihp in range(n_hpo_trials):
    print('Hyperparameter trial %i conv %s fc %s dropout %.4f opt %s, lr %.4f' %
          (ihp, conv_sizes[ihp], fc_sizes[ihp], dropout[ihp], optimizer[ihp], lr[ihp]))
    checkpoint_file = os.path.join(checkpoint_dir, 'model_%i.h5' % ihp)
    result = lv.apply(build_and_train,
                      input_dir, n_train, n_valid,
                      conv_sizes=conv_sizes[ihp], fc_sizes=fc_sizes[ihp],
                      dropout=dropout[ihp], optimizer=optimizer[ihp], lr=lr[ihp],
                      batch_size=batch_size, n_epochs=n_epochs,
                      checkpoint_file=checkpoint_file)
    results.append(result)
```

```
Hyperparameter trial 0 conv [ 64  16 128] fc [128] dropout 0.3234 opt Nadam, lr 0.0100
Hyperparameter trial 1 conv [ 4  8 64] fc [64] dropout 0.6747 opt Adadelta, lr 0.0010
```

Load-balanced scheduling

Launch user-defined training function and arguments

AsyncResult objects can be queried for status, outputs

Distributed HPO with widgets

Notebook widgets can be added to enhance the HPO workflow

Real-time monitoring

- View live status/summaries of HPO training tasks
- Plot detailed live information of select training runs

Enhanced interactivity

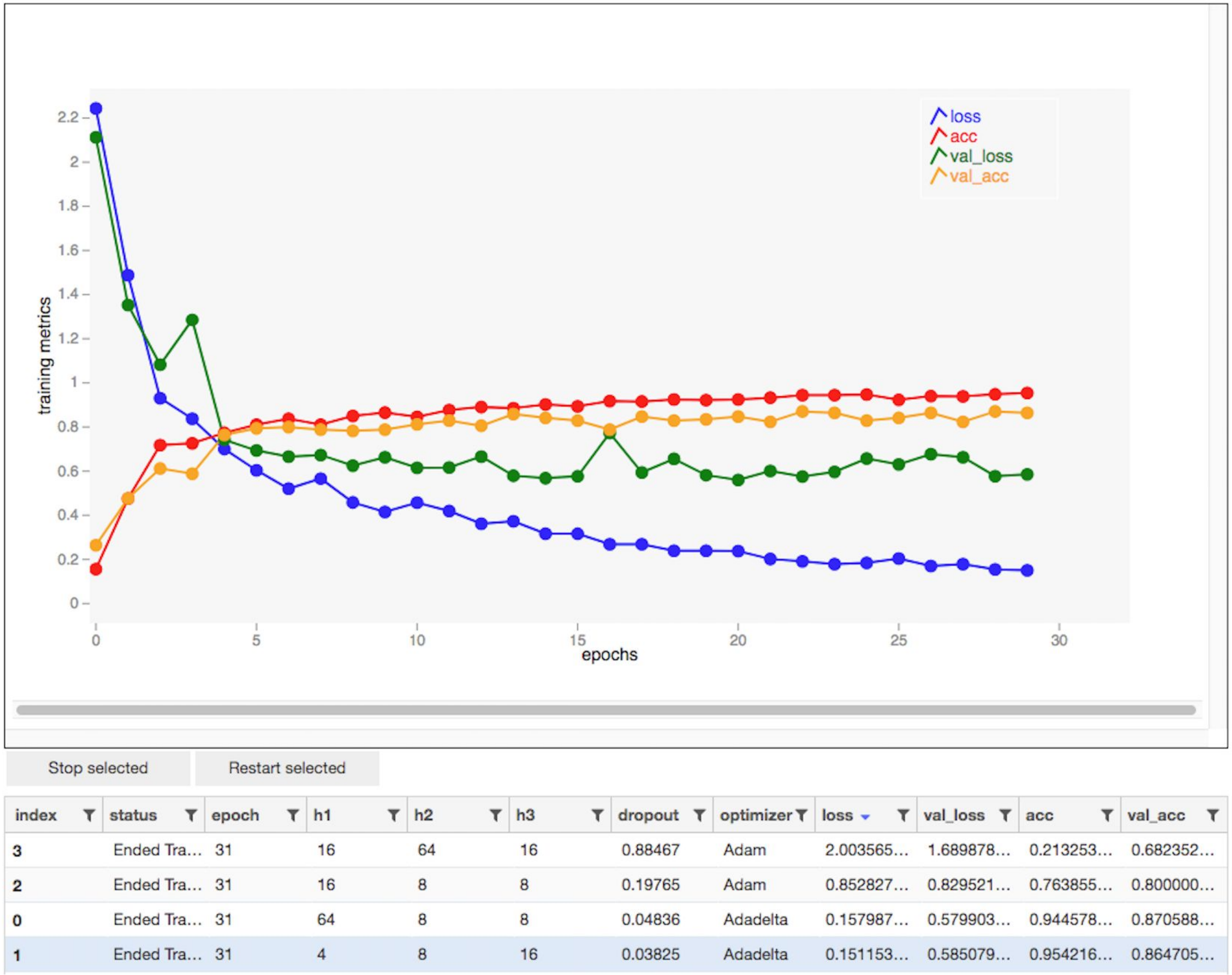
- Select best/worst performing runs
- Do further analysis in notebook
- Modify HP search space
- Start/stop runs

Plots update live

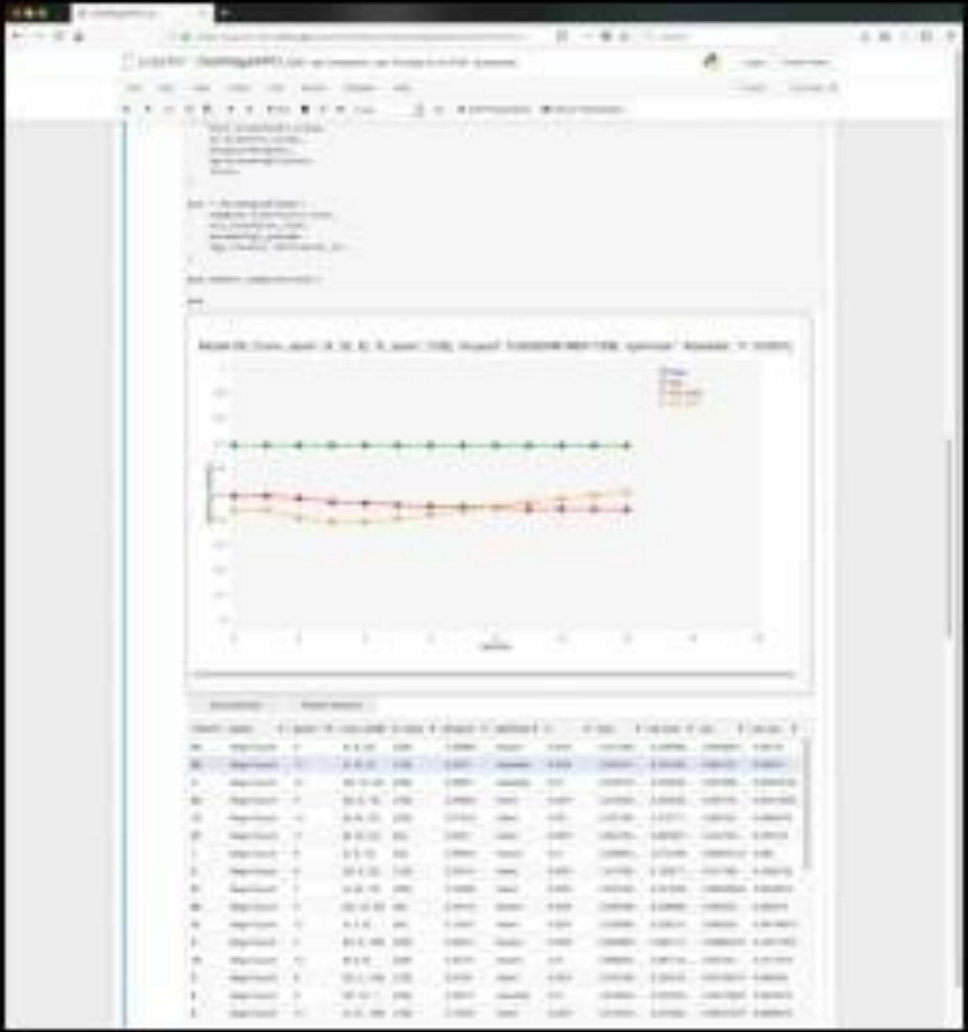
Table shows different configurations:
- Status
- Current loss and accuracy
- Sort

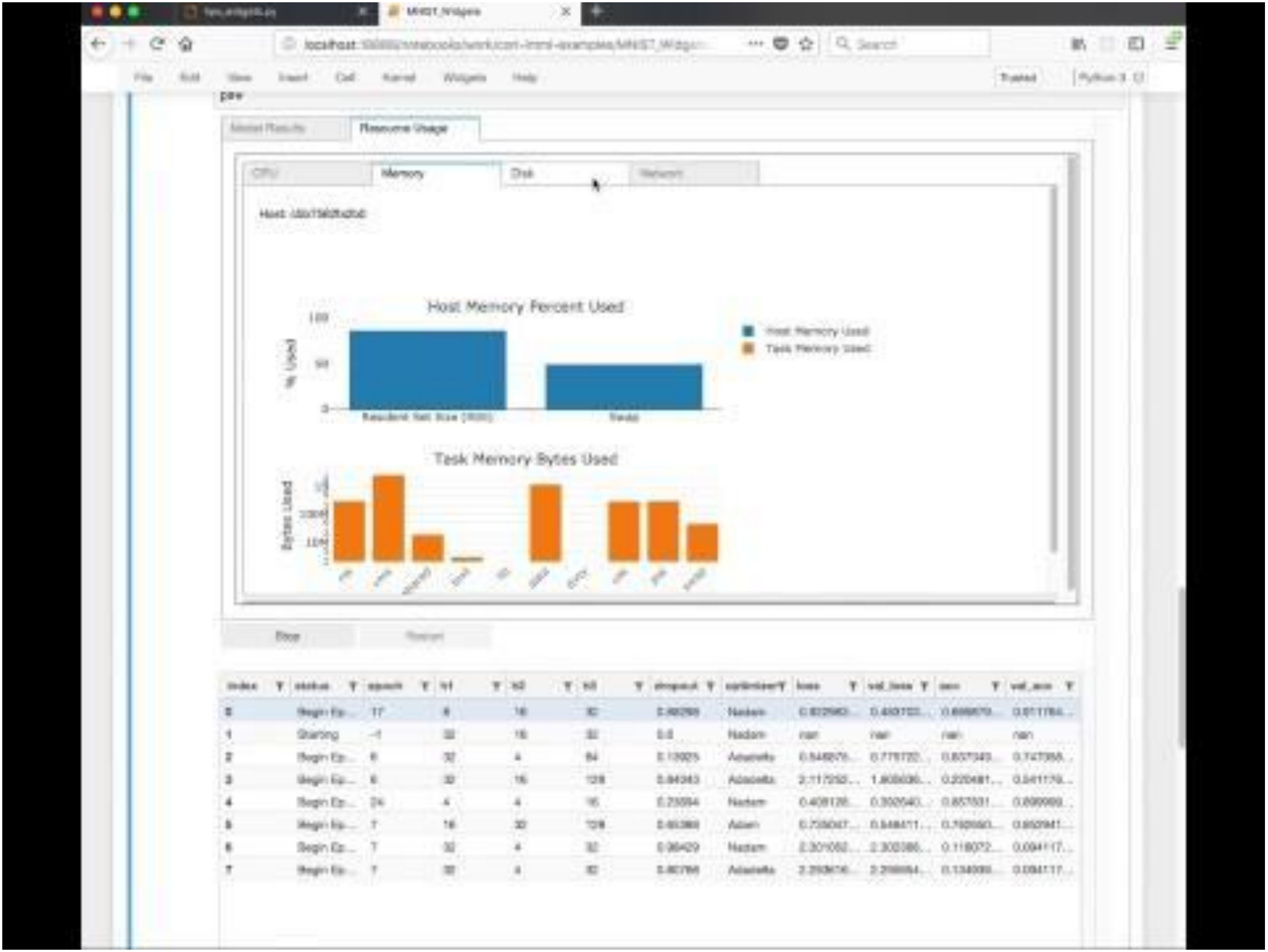
Can add further quantities to plot and interaction buttons

Stop and Restart Tasks



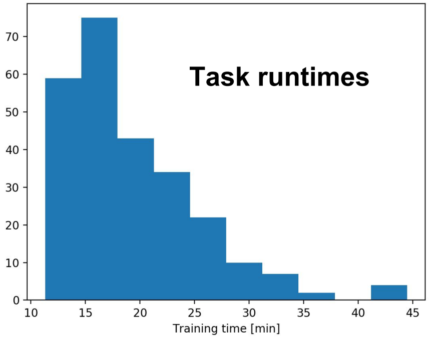
index	status	epoch	h1	h2	h3	dropout	optimizer	loss	val_loss	acc	val_acc
3	Ended Tra...	31	16	64	16	0.88467	Adam	2.003565...	1.689878...	0.213253...	0.682352...
2	Ended Tra...	31	16	8	8	0.19765	Adam	0.852827...	0.829521...	0.763855...	0.800000...
0	Ended Tra...	31	64	8	8	0.04836	Adadelta	0.157987...	0.579903...	0.944578...	0.870588...
1	Ended Tra...	31	4	8	16	0.03825	Adadelta	0.151153...	0.585079...	0.954216...	0.864705...



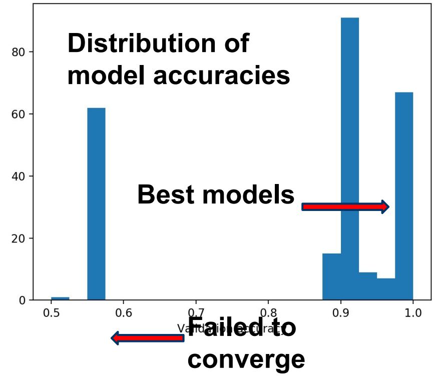

Resource Usage
CPU
Memory
Disk
Host Memory Percent Used
% Used
100
50
0
Host Memory Used
Task Memory Used
Task Memory Bytes Used
Bytes Used
index
status
epoch
h1
h2
h3
dropout
optimizer
loss
val_loss
acc
val_acc

Distributed HPO - Results

For LHC CNN example, process hundreds of HP tasks in <1hr.
Visualize model classification performance and runtimes



Task runtimes

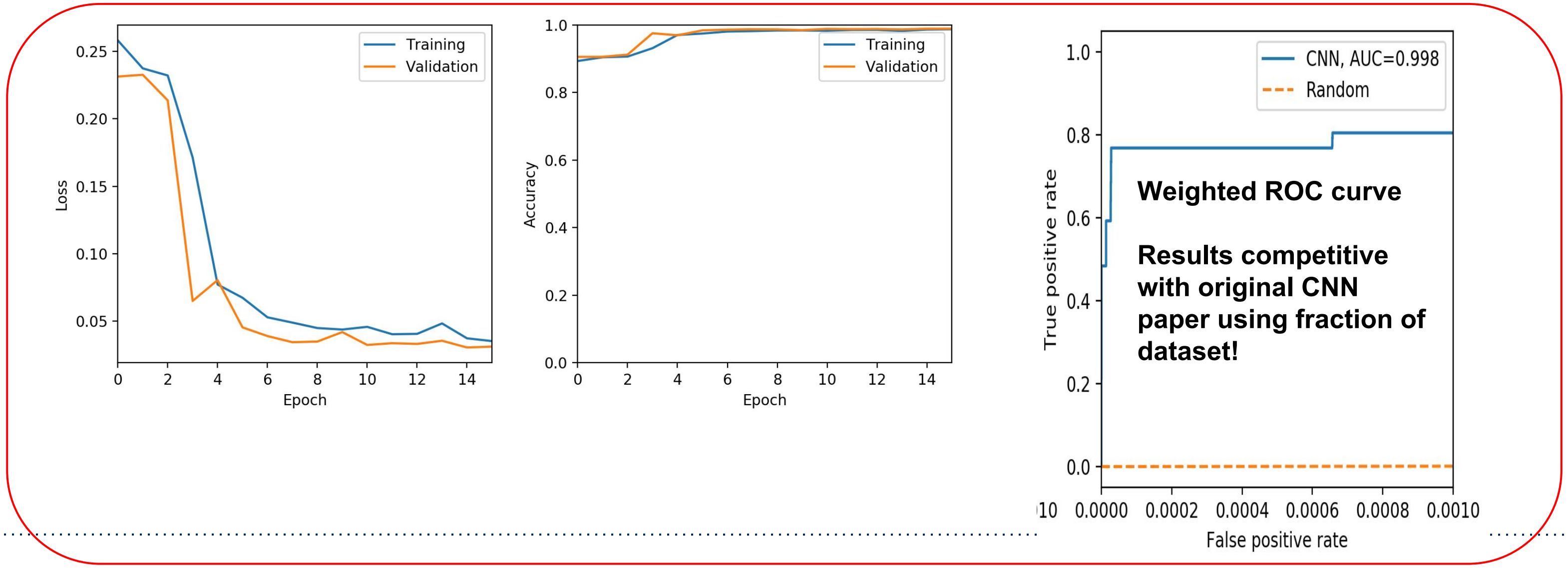


Distribution of model accuracies

Best models

Failed to converge

Best model found:



Weighted ROC curve

Results competitive with original CNN paper using fraction of dataset!

Summary

- Jupyter + Kale + Jupyter Widgets + iPyParallel can give you a powerful platform for iterative, interactive problems on HPC
- Use of Jupyter in deep learning models and hyperparameter optimisation experiments that need distributed HPC resources => clear win for science
- We are developing software and infrastructure for this on Cori at NERSC
- What we’re doing now:
    - Demonstrating and sharing notebook-driven examples for multiple use cases
    - Capturing widgets and code as pluggable modules

Links, Acknowledgements etc.

- Kale:
    - https://github.com/Jupyter-Kale/kale
- Deep Learning Examples:
    - https://github.com/Jupyter-Kale/cori-intml-examples/

- This work was supported by the LBL LDRD program

- Contact:
    - Shreyas Cholia - scholia@lbl.gov
    - Matt Henderson - mhenderson@lbl.gov