

Python for HPC

Andrea Zonca - SDSC

Jupyter Notebook

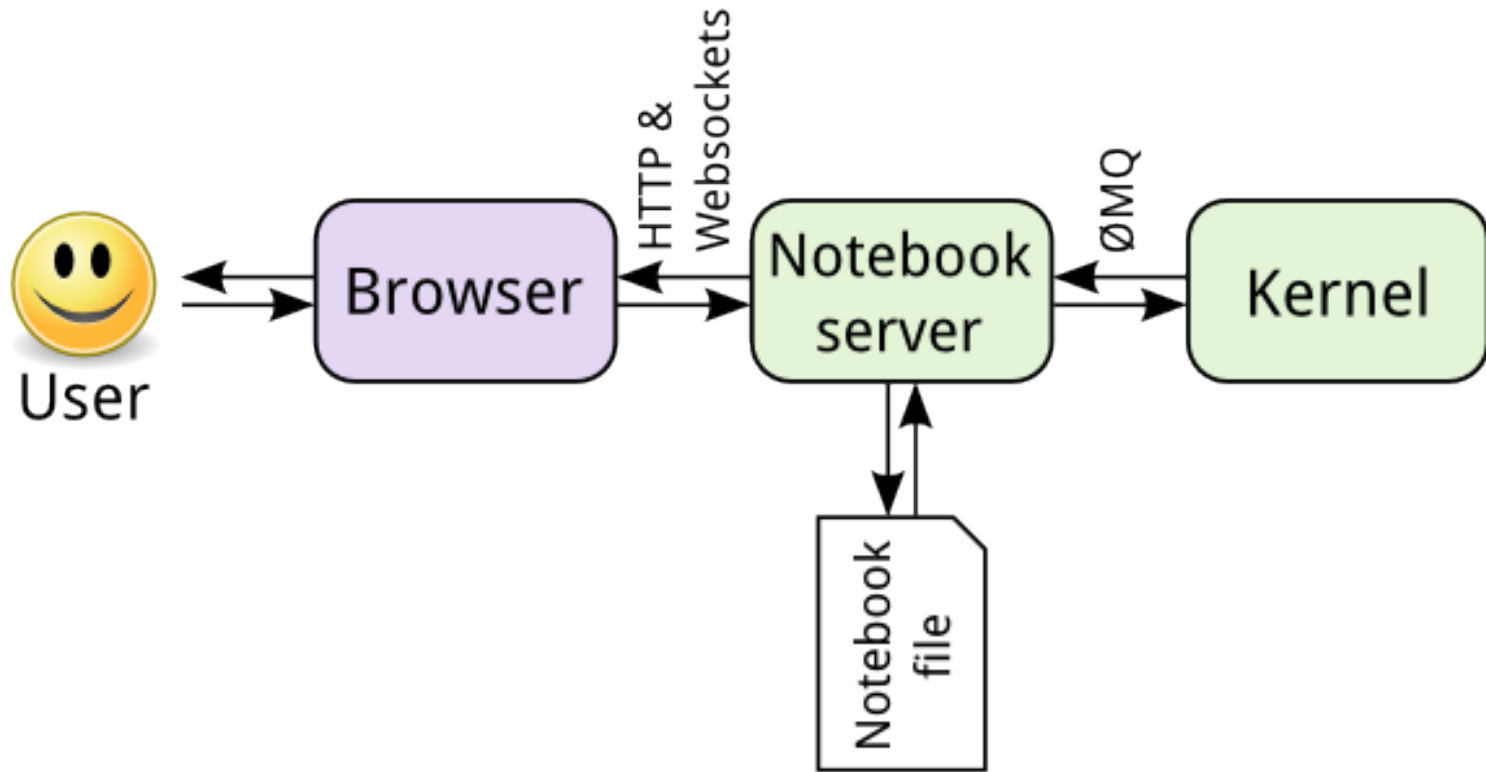
Data exploration in your browser

What is the notebook?

- Browser based interactive console
- Supports multiple sessions in browser tabs
- Each session has a Kernel executing computation
- Saved in JSON format

Notebooks on Nature

<http://www.nature.com/news/interactive-notebooks-sharing-the-code-1.16261>



How the notebook works

Modules on Comet

- module load python
- module load scipy
- add them to .bashrc

Setup on Comet

- ssh to Comet
- `salloc --nodes=1 --tasks-per-node=1 -t 04:00:00 --partition=gpu-shared --res=si2015gpushared --gres=gpu:1`
- `ssh comet-xx-xx`
- `ipython notebook --no-browser --ip="*" #`
better setup config file

ssh-tunneling setup

```
ssh etrainXX@comet.sdsc.edu -L 8888:comet-xx-xx:8888 -f -N
```

Open browser on your laptop and connect to localhost:PORT

New -> Notebook

!hostname

IPython notebook demo

- Python code
- Formatted text
- Equations
- Plots
- Cells execution, cells order
- Clear output

Why the notebook?

- Literate programming: code and explanation together
- Reproducible science: document easily every step
- Easy to share computations: send one single notebook instead of scripts/plots/.doc

ipynb documents

- JSON format
- includes plots in binary format
- easy to convert to .html/.pdf for sharing
- <http://nbviewer.ipython.org>
- Recently rendered automatically on Github

HPC: interactive notebooks

- Analyze large amount of data
- In-situ visualization
- Centralized Python stack
- Check long-running computations
- Prepare and submit batch jobs

Notebooks as scripts

- demo of runipy
 - open and execute `fit_line.ipynb`
 - uncomment cell with `(os.environ)`
 - `white_noise_scale=1000 runipy fit_line.ipynb fit_line_1000.ipynb`
 - open `fit_line_1000.ipynb`, what happened?
- demo of batch submission of SLURM serial runipy jobs using pipes

Hands-on

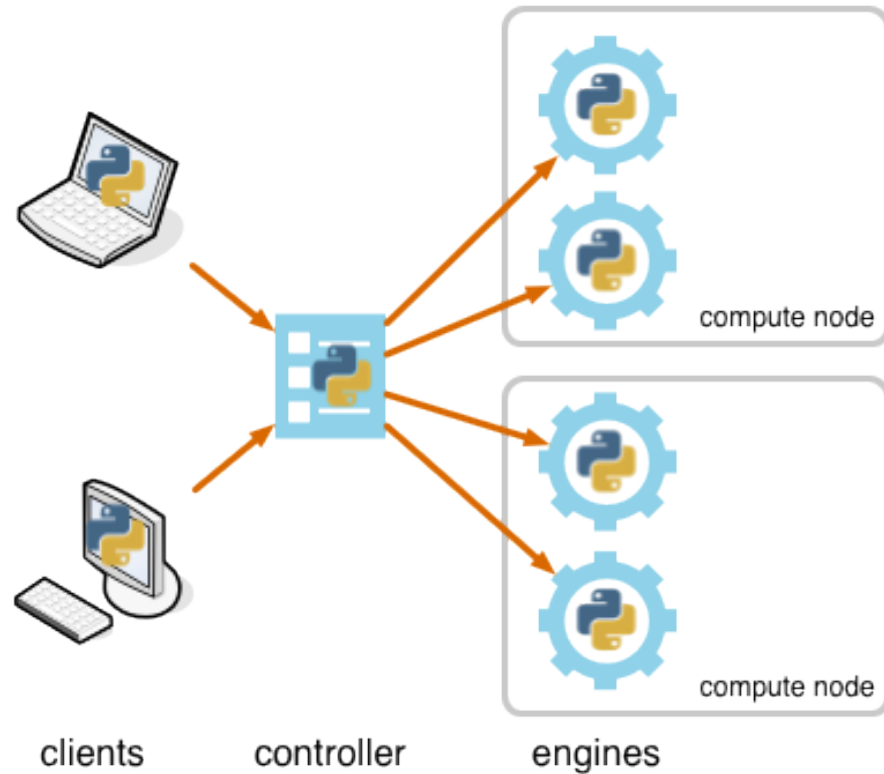
- Open the notebook interactively
- Add saving the plot with `plt.savefig("figurename.png")` in the same cell
- Test with `runipy` on the interactive node
- Rerun the jobs through the queue

IPython parallel

Parallel computing the easy way

IPython parallel

- High-level API for distributed computing with Python
- Engines (Python worker processes) connected to Controller with ZeroMQ
- Client, i.e. user's IPython session, connects to the Controller



independent python kernel

Image by Continuum Analytics

IPython parallel architecture

Functionalities

- Load balanced queue for trivially parallel jobs
- Supports job dependencies
- Direct interface to Engines
- Supports MPI applications, Python or C/C++/Fortran

IPython parallel config

- `ipython profile create --parallel`
- `ipcontroller_config.py`: `c.HubFactory.ip = u'*`
- `ipcluster_config.py`:
 - `c.IPClusterEngines.engine_launcher_class = 'PBSEngineSetLauncher'`
 - `c.PBSEngineSetLauncher.batch_template_file = 'slurm.engine.template'`
 - `c.PBSEngineSetLauncher.submit_command =`

IPython parallel Demo

- Launch cluster with 48 engines:
 - `ipcluster start --n=48`
- Connect with IPython Notebook
- Print ids, hostnames
- Launch demo job and check it runs correctly

Hands-on

- Create a duplicate of `fit_line.ipynb`
- Reformat `fit_line` code into a single function
- Send it to engines for execution within the balanced queue
- Print out the results from the notebook


```
In [1]: from IPython.parallel import Client
```

```
In [2]: c = Client()
```

```
In [3]: view = c[:]
```

```
In [4]: view.activate() # enable magics
```

```
# run the contents of the file on each engine:
```

```
In [5]: view.run('psum.py')
```

```
In [6]: view.scatter('a', np.arange(16, dtype='float'))
```

```
In [7]: view['a']
```

```
Out[7]: [array([ 0.,  1.,  2.,  3.]),  
         array([ 4.,  5.,  6.,  7.]),  
         array([ 8.,  9., 10., 11.]),  
         array([12., 13., 14., 15.])]
```

```
In [7]: %px totalsum = psum(a)
```

```
Parallel execution on engines: [0,1,2,3]
```

```
In [8]: view['totalsum']
```

```
Out[8]: [120.0, 120.0, 120.0, 120.0]
```

Numba

Run code on GPU with Python

JIT compiler for Python

- based on LLVM (compiler infrastructure behind clang, Apple's C++ compiler)
- turns Python code into machine code
- on-the-fly

Numba

```
export NUMBAPRO_NVVM=/usr/local/cuda-7.0  
/nvvm/lib64/libnvvm.so
```

```
export  
NUMBAPRO_LIBDEVICE=/usr/local/cuda-7.0  
/nvvm/libdevice/
```

Interactive GPU node

```
salloc --nodes=1 --tasks-per-node=24 --  
partition=gpu -t 01:00:00
```

```
from numba import jit
from numpy import arange
```

```
# jit decorator tells Numba to compile this function.
```

```
# The argument types will be inferred by Numba when function is called.
```

```
@jit
```

```
def sum2d(arr):
```

```
    M, N = arr.shape
```

```
    result = 0.0
```

```
    for i in range(M):
```

```
        for j in range(N):
```

```
            result += arr[i,j]
```

```
    return result
```

```
a = arange(9).reshape(3,3)
```

```
print(sum2d(a))
```

Numba CPU

run with `%timeit`

increase size of matrix to see performance improvements

Numba GPU

```
from numba import cuda

@cuda.jit
def matmul(A, B, C):
    """Perform square matrix multiplication of  $C = A * B$ 
    """
    i, j = cuda.grid(2)
    if i < C.shape[0] and j < C.shape[1]:
        tmp = 0.
        for k in range(A.shape[1]):
            tmp += A[i, k] * B[k, j]
        C[i, j] = tmp

import numpy as np
shape = (5,5)
a = np.ones(shape)
b = np.ones(shape) * 4
c = np.zeros(shape)
matmul[1,(16,16)](a,b,c)
print(c)
```

Hands-on

- create a loop that runs `matmul` with different matrix sizes
- compare with `np.dot`
- range from `20x20` to `10000x10000`
- plot timing

Advanced CUDA

Tiled matrix multiplication to exploit GPU fast local memory:

<http://numba.pydata.org/numba-doc/0.20.0/cuda/examples.html#cuda-matmul>

PyTrilinos

Distributed linear algebra with Python

Distributed linear algebra

Large complete C++ packages with Python support:

- PETSC, petsc4py
- Trilinos, PyTrilinos

Both use C++ for MPI communication and LAPACK/BLAS for local computing

Both subclass numpy arrays

PyTrilinos example

See `pytrilinos.ipynb`