Parsl: A Parallel Scripting Library for Python

Kyle Chard (chard@uchicago.edu)
Yadu Babuji, Anna Woodard, Ian Foster, Dan Katz, Mike Wilde, Justin Wozniak

http://parsl-project.org
Parsl: Parallel scripting in Python

Annotate functions to make Parsl apps
- Bash apps call external applications
- Python apps call Python functions

Apps run concurrently respecting data dependencies via futures. Natural parallel programming!

Parsl scripts are independent of where they run. Write once run anywhere!
Futures

Futures are a proxy for a result that is unknown (e.g., from asynchronous execution)

AppFuture
- Invocation of apps returns a future for managing execution and controlling the workflow

DataFuture
- Represents data produced by an app
- Enables construction of dataflow by connecting apps with DataFutures
- Parsl monitors to ensure files are created and that they are passed to dependent apps

```python
# App that sleeps and then returns hello world
@App('python', dfk)
def hello ():
    import time
    time.sleep(5)
    return 'Hello World!'

app_future = hello()

# Check if the app_future is resolved
print ('Done: %s' % app_future.done())

# Print the result of the app_future. Note: this call will block and wait for the future to resolve
print ('Result: %s' % app_future.result())
print ('Done: %s' % app_future.done())
```
Parsl scripts are execution provider and execution model independent

The same script can be run locally, on grids, clouds, or supercomputers

- Works directly with the scheduler (no HTC-like setup)

A single script may use many execution providers

- Parsl builds on libsubmit
  - https://github.com/Parsl/libsubmit
  - Local, Cloud (AWS, Azure, private), Slurm, Torque, Condor, Cobalt

A single script may use various execution models

- Threads, pilot jobs, extreme scale (Swift/T)

Configuration file describes how to use resources
Interactive supercomputing with Jupyter notebooks

- Parsl can be used from within a Jupyter notebook
- Built-in visualization and management
  - Parsl graph
  - Status and debugging
- Transparent pass through of authentication tokens in JupyterHub
Parsl tutorial

Running the tutorial online:

– Online notebooks: http://try.parsl-project.org
– Binder: https://mybinder.org/v2/gh/Parsl/parsl-tutorial/master

Running the tutorial on Blue Waters

– Set up Parsl and download tutorial
  • module load bwpy
  • pip install --user parsl
  • git clone https://github.com/Parsl/parsl-tutorial
– Execution
  • Download code and run in terminal
  • **Execute notebook on Blue Waters**
    – https://bluewaters.ncsa.illinois.edu/pythonnotebooks
  • Execute notebook remotely (e.g., laptop) using Blue Waters
Large-scale applications using Parsl

A. Machine learning to predict stopping power in materials
B. Protein and biomolecule structure and interaction
C. Information extraction to discovery facts in publications
D. Materials science at the Advanced Photon Source
E. Cosmic ray showers as part of QuarkNet
F. Weak lensing using sky surveys
G. Machine learning and data analytics
Conclusion: parallel workflow scripting is practical, productive, and necessary, at a broad range of scales

Parsl takes a highly successful parallel scripting model and brings it to Python

- No porting of existing scripts to other languages
- Support for both Python and external app functions

Already applied to numerous MTC and HPC application domains

- Attractive for data-intensive applications
- Hybrid programming models

Deep integration with growing ecosystem:

- Globus, Python, Jupyter, workflow library, ...

Workflow through implicitly parallel dataflow is productive for applications and systems at many scales, including on highest-end system
Parsl Resources

- Getting started
  - http://try-parsl.parsl-project.org

- Parsl tutorial
  - https://github.com/Parsl/parsl-tutorial

- Documentation
Questions?

http://parsl-project.org

Try Parsl: http://try.parsl-project.org