INTERACTIVE, SCALABLE, REPRODUCIBLE DATA ANALYSIS WITH CONTAINERS, JUPYTER, AND PARSL

ANNA WOODARD*, YADU BABUJI, KYLE CHARD, IAN FOSTER, DANIEL S. KATZ, MIKE WILDE, JUSTIN WOZNIAK

*annawoodard@uchicago.edu
Parsl is a Python-based workflow system.
"TRADITIONAL" ANALYSIS BATCH SUBMISSION IN HEP

```python
for iList in listOFSamples:
    condorJobFile = open("dilBatch.submit", "w")
    condorJobFile.write("universe = vanilla\n"+"executable = runPlotsCondor.csh\n")
    condorJobFile.write("Zmask = %s\n" % iZmask)
    condorJobFile.write("Label = %s\n" % jobLabel)
    condorJobFile.write("List = %s\n" % iList)
    condorJobFile.write("JES = %s\n" % jesChoice)
    condorJobFile.write("JER = %s\n" % jerrChoice)
    condorJobFile.write("PV = %s\n" % iPV)
    condorJobFile.write("Charge = %s\n" % iCharge)
    condorJobFile.write("arguments = $\{List\} $\{Zmask\} $\{Label\} $\{PV\} $\{Charge\}\n")
    condorJobFile.write("output = batchBEAN/condorLogs/condor_\{List\}_\{Process\}.stdout\n")
    condorJobFile.write("error = batchBEAN/condorLogs/condor_\{List\}_\{Process\}.stderr\n")
    condorJobFile.write("queue 1\n")
```

assumes fixed execution resources

manually produce submit script for specific Tier 2/ Tier 3

dependencies resolved by manually running steps sequentially

processing step A step B step C time
HOW IS PARSL DIFFERENT?
ParSL Basics

- Pure python; easy installation
- Rather than define code/input/output mapping externally, the user annotates functions to make ParSl apps
  - Bash apps call external applications
  - Python apps call Python functions
- Apps return “futures”: a proxy for a result that may not yet be available

```
@python_app
def hello():
    return 'Hello world!'

print(hello().result())

Hello world!

@bash_app
def echo_hello(stdout='hello.stdout'):
    return 'echo "Hello world!"

echo_hello().result()
with open('hello.stdout') as f:
    print(f.read())

Hello world!
```
Apps run concurrently, respecting data dependencies via futures. Implicit parallel programming!

Dynamic: apps can create apps! Apps can be recursive!
Parsl scripts are independent of where they run. Write once, run the same script locally, on grids, clouds, or supercomputers!

A single script may concurrently use separate pools of resources, with different execution models.

Supported providers:
AWS, Azure, Google Cloud, Slurm, Torque, HTCondor, Cobalt

*Note the format of this configuration will be supported in Parsl 0.6, being released this week.
PARSLS FEATURES

- Apps can be shared as libraries
- Elasticity: resources used are scaled up and down according to demand automatically
- App caching and checkpointing: re-use results if app is called with the same inputs (record of inputs and outputs = provenance capture!)
- Workers can be launched in Docker containers (re-used for multiple apps); Docker/Shifter/Singularity/etc containers can be used with wrappers for per-app containerization
- Data transfer: Globus, HTTP, FTP

```python
file = File(globus://endpoint/path/file)
```
PARSL IS ALREADY BEING USED IN A VARIETY OF DOMAINS.

WHY NOT ADD HEP?
# WHAT DO HEP TASKS NEED?

<table>
<thead>
<tr>
<th>requirement</th>
<th>solution used</th>
</tr>
</thead>
<tbody>
<tr>
<td>specific OS / run environment</td>
<td>vc3-builder (starts Singularity containers if they are needed) [1]</td>
</tr>
<tr>
<td>CVMFS mounted in userspace</td>
<td>Parrot (via vc3-builder) [2]</td>
</tr>
<tr>
<td>HEP software stack + user code</td>
<td>sandbox wrapper [3]</td>
</tr>
</tbody>
</table>

[1] [http://virtualclusters.org](http://virtualclusters.org)
[2] [https://ccl.cse.nd.edu/software/parrot/](https://ccl.cse.nd.edu/software/parrot/)
[3] [https://github.com/NDCMS/lobster](https://github.com/NDCMS/lobster)
EXAMPLE IMPLEMENTATION FOR CMSSW+PARSL

CMS-specific decorator
(this is an example [1];
could be modified for other experiments)

Parsl task
wrapper (user code)
container (OS, CVMFS)
worker

Parsl task
wrapper (user code)
container (OS, CVMFS)
worker

vc3-builder cache

[1] https://github.com/annawoodard/parslcms
WHY BOTHER WITH NOTEBOOKS?

- Traditional HEP analysis paradigm: code, results, and documentation are separate. Hard to keep synchronized!
  - Notebooks allow these to be combined into a single narrative
- Web interface facilitates sharing
- Interactive plotting
- Native caching: fast, iterative development
- Jupyter Lab: text editors, terminals, data file viewers, and other custom components side by side with notebooks in a tabbed work area!
- Barriers to notebook adoption in HEP: complex software stacks, use of distributed computing
PARSL + NOTEBOOKS FOR HEP

Real-world example [1] implementing part of my dissertation workflow in Parsl

SUMMARY

- Parsl’s implicit dataflow model allows for simple expression of complex dependencies
- In Parsl, code is separate from the specification of computing resources: this makes Parsl scripts portable and scalable
- Parsl has a number of useful features: app caching, elasticity, container support, data transfer, and more
- To extend Parsl for use in HEP, an example has been shown which wraps apps in a Singularity container with CVMFS mounted via VC3-builder, and the CMSSW user code set up via sandboxing. The workflow is orchestrated via a Jupyter notebook, which facilitates easy sharing and documentation, and fast iterative development.
STAY IN TOUCH!

Parsl

http://parsl-project.org
BACKUP
“TRADITIONAL” GRID SUBMISSION IN HEP

user explicitly defines:

- input dataset
- method for splitting dataset into chunks
- code to execute on each chunk

```python
from CRABClient.UserUtilities import config
config = config()
config.Data.inputDataset = '/SingleMu/Run2012B-13Jul2012-v1/AOD'
config.Data.splitting = 'LumiBased'
config.JobType.psetName = 'pset_tutorial_analysis.py'
```