Parsl: A Python-based Parallel Scripting Library

http://parsl-project.org

Yadu Babuji[*], Kyle Chard[⚑], Ben Clifford[*], Ian Foster[*⚑], Daniel S. Katz°, Lukasz Lacinski[*⚑], Zhuozhao Li[*], Connor Pigg°, Michael Wilde*[⚐], Anna Woodard[*], Justin M. Wozniak[*]*

[*]University of Chicago & Argonne National Laboratory; [⚑]Globus; [⚐]Parallel Works
°National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

Goals

- Easy to write Python workflows that glue together external programs and Python functions
- Easy to run in parallel on diverse resources
- Easy to install:

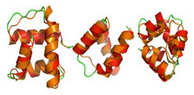
```
pip install parsl
```

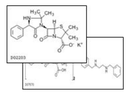
- Open source (Apache 2.0 license)
- Open community

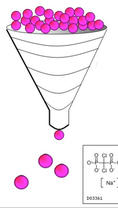
Scientific Workflows

HPC/HTC workflow

O(10) proteins X O(100K) drug candidates

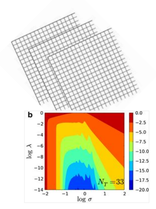




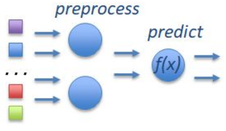


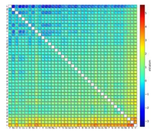
=1M docking tasks, hundreds of MD models to find candidates for experiments

Machine learning workflow



O(Ms) of data used to train a model

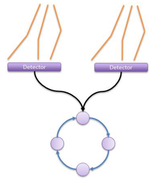




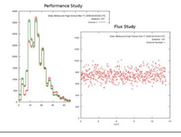
Model applied to O(Ms) of possible materials

Interactive workflow

O(1M) cosmic ray events



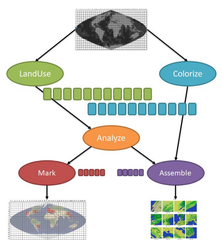
Iterative collection, curation, analysis, visualization



Write once, run anywhere

On clouds, clusters, supercomputers

Parsl scripts are independent of the execution environment. A single script can be executed on one or more execution resources without modifying the script.



Implicit dataflow

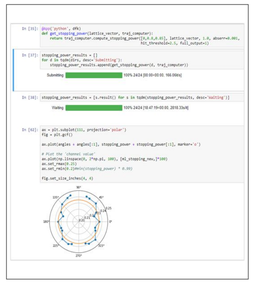
Apps execute concurrently while respecting data dependencies

Parsl creates a dynamic graph of tasks and their data dependencies. Tasks are only executed when their dependencies are met.

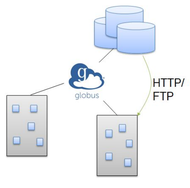
Scalable Jupyter notebooks

Easily manage execution across distributed resources

Parsl works seamlessly with Jupyter notebooks allowing apps within a notebook to be executed in parallel and on remote resources.



```
parsl_file =
  File(globus://EP/path/file)
```



Automated data movement

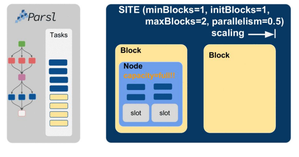
Implicit wide area staging

Parsl handles the complexity of ensuring data is in the right place at the right time for computation.

Execution management

Handles failures and elasticity

Parsl uses checkpointing and automatic retries as a resilience mechanism to handle failures. Parsl apps can be containers in resource pools that grow and shrink elasticity as needed.



Configuration: Use arbitrary resource(s)

```
Comet_config = Config(
    executors=[
        IPyParallelExecutor(
            label='comet_ipp_multinode',
            provider=SlurmProvider(
                'compute'
        ))])

parsl.load(Comet_config)
```

App definition: Run Python and bash apps

```
@bash_app
def generate(outputs=[]):
  # return a random number from 1 to 10
  return "echo $(( ( RANDOM % 10) + 1 )) &> {outputs[0]}"

@python_app
def total(inputs=[]):
  total = 0
  for i in inputs:
      with open(i, 'r') as f:
            total += sum([int(line) for line in f])
  return total
```

Execution: Transparent parallelization based on data dependencies

```
# Create 5 files with random #s
output_files = []
for i in range (5):
   output_files.append(generate(outputs=['r%s.txt' % i]))

# Calculate the sum of the random numbers
t = total(inputs=[i.outputs[0] for i in output_files])
print (t.result())
```

Parsl is supported by the National Science Foundation under awards 1550476, 1550475, 1550528, 1550562, 1550588