Scalable Parallel Scripting in the Cloud

Ben Glick[1], Yadu Babuji[2] (Advisor), Kyle Chard[2] (Advisor)

[1] University of Chicago/Lewis & Clark College, [2] Computation Institute, University of Chicago/Argonne National Lab {glick,chard,yadunand}@uchicago.edu

Abstract

- **Parsl is a Python-based parallel scripting library that enables creation of implicitly parallel, dataflow processing**
- **Parsl abstracts different execution environments enabling the same script to run on multicore computers to supercomputers**
- **In this poster we describe the development of a new execution provider designed to support Amazon Web Services (AWS) and Microsoft Azure**
- **These extensions allow researchers to easily execute implicitly parallel dataflow scripts using elastic cloud resources.**

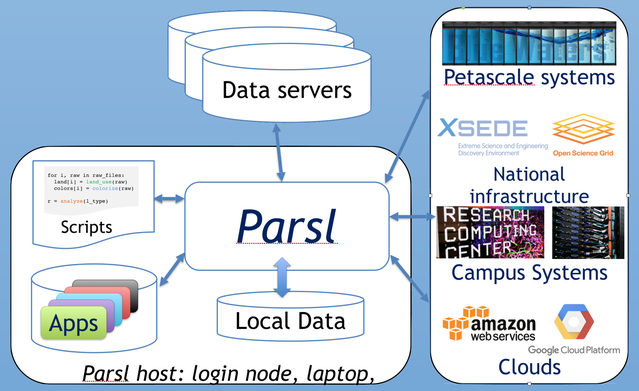
Parsl

Parsl's pythonic base makes it easy to learn and use, while its configurable choice of backends makes it both powerful and flexible

Parsl is parallel: it runs multiple programs concurrently as soon as their inputs are available, reducing the need for complex parallel programming

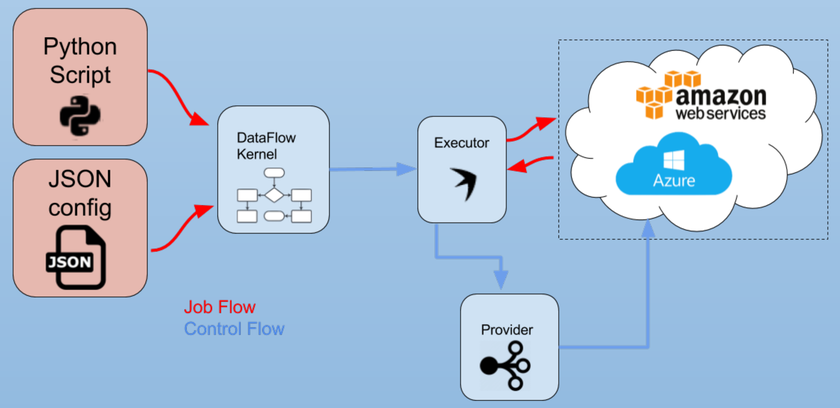
Parsl is fast: it can run a million programs, thousands at a time, launching thousands per second

Parsl is easy: short, simple scripts can do large-scale work. The same script runs on multicore computers, clusters, grids, clouds, and supercomputers



Architecture

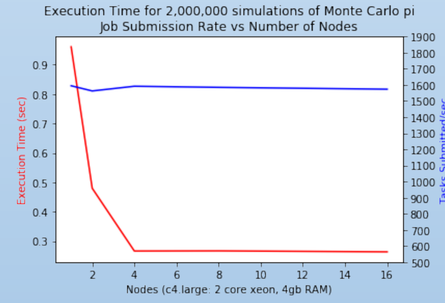
- **Parsl uses an execution provider to provide execution resources for the job to use**
- **We have created new execution providers for AWS and Azure that take advantage of elastically scalable execution infrastructure**
- **We interact with AWS through the python library Boto3 and with Azure through the Azure SDK for Python**
- **On startup of an instance, we execute a pilot job, which connects the instance to an IPython Parallel cluster from which a parsl executor can submit more jobs**



Code Sample

```
provider = parsl.execution_provider.aws.EC2Provider('path/to/config')
executor = parsl.executors.ipp.IPyParallelExecutor()
dfk = DataFlowKernel(executor)
@App('python', dfk)
def parsl_app(n):
    arbitrary_function()
    return 0
provider.scale_out(16)
#parsl will automatically run the code on all workers it can find
results = [parsl_app(n).result() for n in range(1,100,1)]
```

Evaluation



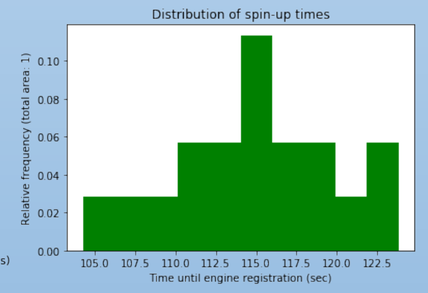
Parsl's AWS provider can launch more than 1500 tasks per second regardless of cluster size

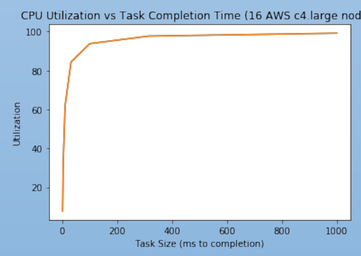
Parsl is able to continue to launch tasks quickly even afte thousands of tasks have been submitted

Monte carlo simulation of pi demonstrates scalability as nodes are added

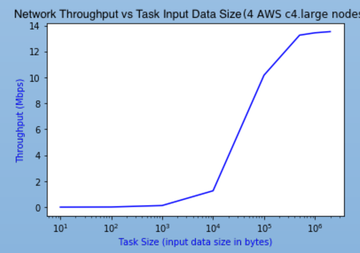
Parsl can instantiate , configure, and use instances rapidly

Less than 2 minutes after a scale_out request is submitted, the new engines are ready to use





As job execution time increases, CPU utilization rapidly approaches 100%



As job input data size increases, overhead becomes negligible and network thoughput increases

Try Parsl

- **More information: (http://parsl-project.org)**
- Live Jupyter Notebooks (http://try.parsl-project.org)