Python-based workflows with Parsl

Daniel S. Katz, dskatz@illinois.edu, d.katz@ieee.org, @danielskatz
Assistant Director for Scientific Software & Applications, NCSA
Research Associate Professor, CS, ECE, iSchool
Python-based workflows parallelism with Parsl

Daniel S. Katz, dskatz@illinois.edu, d.katz@ieee.org, @danielskatz
Assistant Director for Scientific Software & Applications, NCSA
Research Associate Professor, CS, ECE, iSchool
An increasingly common story…

- I’m developing an application and I need to link together external tools + functions
  - (where each tool is dependent on data from the previous tool)
- I have a notebook that does X and I need to run it on a cloud, cluster, supercomputer
- I need to run my analysis using a range of local and distributed datasets
- I like Python
- ...
- And I want to do this in an interactive environment
Annotate functions to make Parsl apps
- Python apps call Python functions
- Bash apps call external applications

Apps return “futures”: proxies for results that might not be available yet

Apps run concurrently respecting data dependencies - Natural parallel programming!

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

Based on ideas in Swift, completely reimplemented
Parsl in action: dynamic dataflow execution

- The same script can be run locally, on grids, clouds, or supercomputers (or multiple)
  - Works directly with the scheduler (no HTC-like setup)
- Containers can be used for per-app execution or repeated invocation of the same app
- Currently supported execution providers:
  - Local, Cloud (AWS, private), Slurm, Torque, Condor, Cobalt
A variety of execution models

- High throughput (HTEX)
  - General-purpose pilot job model, 2k nodes, O(M) tasks
- Extreme scale (EXEX)
  - MPI-based pilot jobs, >250k workers on 8k Blue Waters nodes
- Low latency (LLEX)
  - Using ZeroMQ, <5 ms
- New execution models can be added
Transparent (wide area) data management

- Implicit data movement to/from repositories, laptops, supercomputers, …
- Globus for third-party, high performance and reliable data transfer
  - Support for site-specific DTNs
- HTTP/FTP direct data download/upload
- Complements node-specific staging and caching models

```python
parsl_file = File(globus://EP/path/file)
```
Parsl feature summary

- Parsl’s implicit dataflow model allows intuitively expressing parallelism that is then made possible at execution time via an executor
  - Expressed directly in Python
  - Can be used to implement a range of workflow models
- Parsl integrates with the scientific ecosystem
  - Development and execution of scalable applications in Jupyter
  - Use of common SciPy libraries
  - Integration with Globus
- In Parsl, code is separate from the specification of computing resources and data location: this makes Parsl scripts portable and scalable
- Parsl has a number of other important features:
  - App caching, checkpointing, elasticity, container support, data transfer, and more
Opportunities for collaboration

• Generally be able to interact and interoperate with other parallel programming/scripting tools
• Given that exact sequence of tasks is unknown until runtime, how should we store provenance and support reproducibility
• Define & implement common description of tasks for multiple scripting/workflow systems
• Define & implement common interfaces to classes of schedulers/systems
• Define & implement common endpoints for specific systems
• Define & implement common AA methods
• Understand how to find & store metadata for citation of programs, including documentation of authorship and assignment of credit
Acknowledgements

- Parsl (http://parsl-project.org) team