Distributed Dataflows with Parsl + funcX: a FaaS based Approach

Zhuozhao Li
Assistant Professor
Southern University of Science and Technology
The needs for distributed dataflows

Dependency-based tasks

Phenomenal growth of data volumes and velocities

Distributed data and compute
Parsl: Enabling composition and dynamic dataflow graph

```python
@python_app
def pi(num_points):
    from random import random
    inside = 0
    for i in range(num_points):
        x, y = random(), random()  # Drop a random point in the box.
        if x**2 + y**2 < 1:  # Count points within the circle.
            inside += 1
    return (inside*4 / num_points)

# App that computes the mean of three values
@python_app
def mean(a, b, c):
    return (a + b + c) / 3

# Estimate three values for pi
a, b, c = pi(10**6), pi(10**6), pi(10**6)

# Compute the mean of the three estimates
mean_pi = mean(a, b, c)

# Print the results
print("Average: {:.5f}".format(mean_pi.result()))
```
Distributed endpoint model

- Lightweight agent deployed by users
- Dynamically provisions resources, deploys containers, and executes functions

Turn any machine into a function serving endpoint
A new **funcX executor** for Parsl (prototype)

Parsl: Python Pervasive Scripting Library

**funcX Service**

Endpoint 1

......

Endpoint N

Programming interface

Execution engine
A new **funcX executor** for Parsl (prototype)

```python
from parsl.executors import FuncXExecutor

fx_config = Config(
    executors=[
        FuncXExecutor(
            Label="funcX",
            # worker_debug=True,
            endpoints=['870b1d5d-28b0-4962-877f-886d96d4d785'],
        ),
    ],
)

parsl.load(fx_config)
```
Task distribution across endpoints

- Parsl: Python Pervasive Scripting Library
- Task Distribution
- funcX Service
- Endpoint 1
- ......
- Endpoint N
Task distribution across endpoints

• Allow distributing tasks to specific endpoints manually
  o Decorators
    @python_app(..., \parsl\_resource\_specification={"endpoints": [ep1, ep2]})
    def foo():
      o On task invocations
    foo(*args, **kwargs, \parsl\_resource\_specification={"endpoints": [ep1, ep2]})

• Advanced topics
  o Automatically and transparently distribute tasks to their most appropriate endpoints
  o E.g., where data is located, resources are available, or compute is the most efficient
  o Extensible to customized task distributing algorithms
Task and endpoint status reports are needed!

• Endpoint status
  o Liveness
  o Available workers
  o Walltime
  o Queue length
  o etc.

• Task status
  o Task utilization
  o Task completion time
  o Data size

def get_endpoint_status(self, endpoint_uuid):
    """Get the status reports for an endpoint.
    Returns
    -------
    dict
    The details of the endpoint's stats
    """

It is extremely challenging to report and store task status across distributed endpoints and at scale!
Data management

• Inter-endpoint transfers
  o E.g., Globus

• Intra-endpoint transfers
  o E.g., Shared file system
  o In-memory store, e.g., Redis clusters
  o Other approaches, e.g., RDMA

• Data proxy --- transparent, uniform interface
Managing environments across endpoints

• **Lightweight function monitor** *(IPDPS’21, collaborative work with Douglas Thain’s lab at ND)*
  - Automatically **detect** the dependencies of functions
  - **Package** conda environments
  - **Distribute** environments to workers

• **Alternatives**: E.g., **container service** for funcX
  - Dynamically build containers for funcX functions for different endpoints (e.g., Docker, singularity)
Distributed Dataflows with Parsl + funcX

- Parsl: Python Pervasive Scripting Library
- Task distribution
- funcX Service
- Environment Management
- Endpoint 1
- ......
- Endpoint N
- Data Management
Benefits

• Enable one to easily compose a distributed dataflow across different endpoints, without worrying too much about the task distribution

• Compute on cluster resources from one’s laptop

• Portable dataflows

• The potential to use distributed computing resources to avoid high queuing time of large jobs

• More...
Thanks!

Questions!
Suggestions!
Potential use cases!