

Tracking File Provenance with Parsl



Doug Friedel

National Center for Supercomputing Applications

University of Illinois at Urbana-Champaign



National Center for
Supercomputing Applications

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN



File Provenance

What is file provenance?

- When a file was created
- How a file was created
 - By what function
 - What input arguments were used
 - What environment was used (e.g. conda packages, env vars)
- What other functions used the file

File provenance gives you the information needed to recreate the file exactly, and to know the entire history of a file within a workflow.



What Use is File Provenance?

Take an example:

- You have a workflow
 - Dozens of tasks
 - Maybe it loops in time steps
 - It produces numerous files
 - You want to run it many times with different parameters
- When looking at the results you want to know how a specific file was produced.
- If you kept meticulous notes you could probably figure it out, but that is a lot of work.
- This is where file provenance comes into play.



File Provenance Tracking in Parsl

How do we track file provenance?

- Utilized *Parsl's* existing monitoring framework to:
 - Identify files that were used as inputs and/or outputs
 - Capture information (size, timestamps, etc.) about each file
 - Capture what task created the file
 - Capture what tasks used each file
 - Capture the input arguments to each task
 - Capture the *Parsl* execution environment (e.g. `worker_init` from *Provider*)

The existing *Parsl* monitoring visualization tool was modified to be able to view the provenance information via a web interface.



Using File Provenance

Using file provenance is straight forward

- Just add to your config

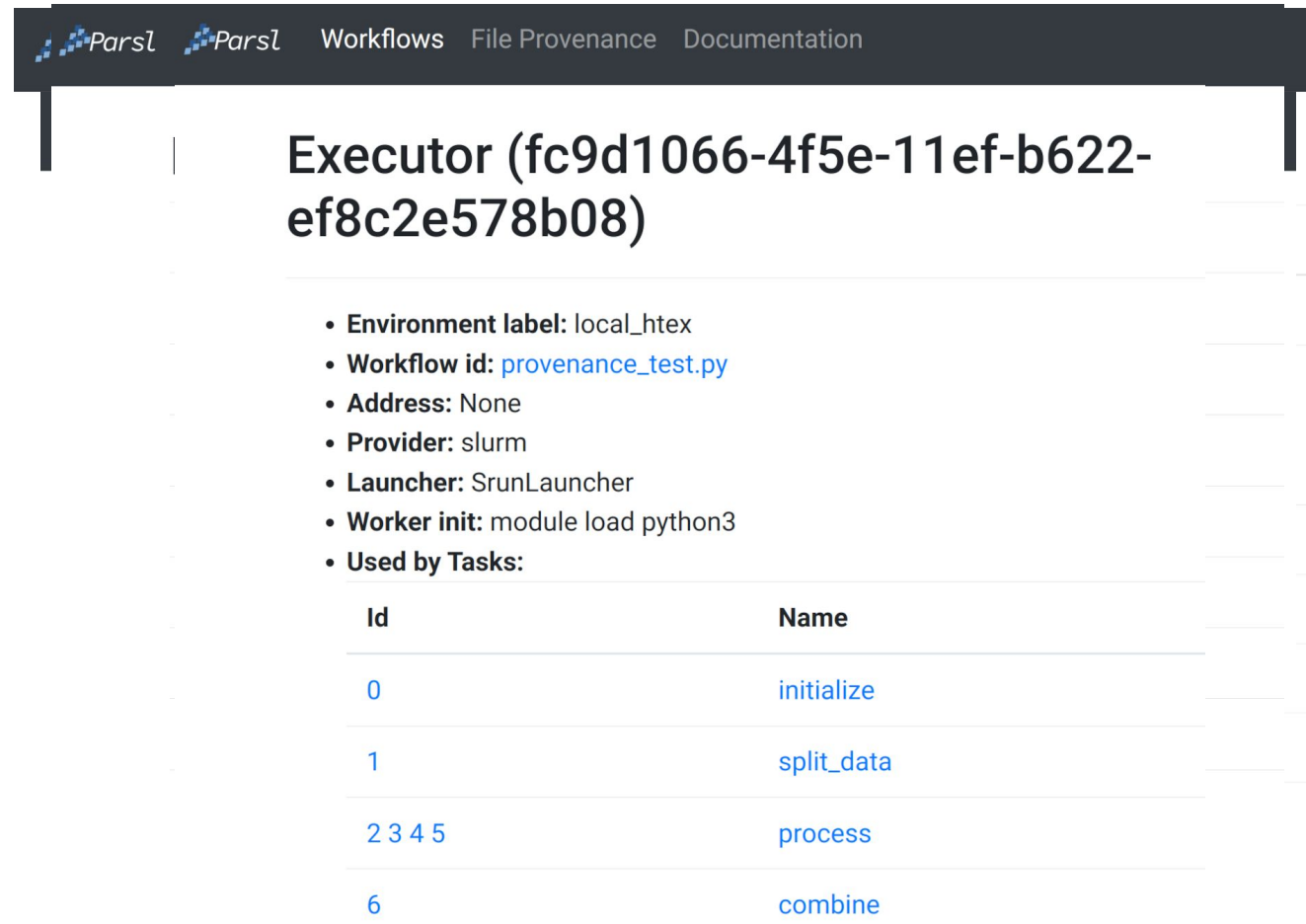
```
from parsl.monitoring.monitoring import MonitoringHub

config = Config(executors=[...],
                Monitoring = MonitoringHub(hub_address=address_by_hostname(),
                                           hub_port=55055,
                                           monitoring_debug=True,
                                           resource_monitoring_enabled=True,
                                           resource_monitoring_interval=1,
                                           capture_file_provenance=True
                                           )
                )
```

Visualizing the Data

Visualizing the output of the file provenance tracking is straight forward

- It is incorporated in the `parsl-visualize` script from the monitoring framework



The screenshot shows a web interface for the Parsl monitoring framework. At the top, there is a navigation bar with the Parsl logo and links for 'Workflows', 'File Provenance', and 'Documentation'. The main content area displays details for an executor with ID 'fc9d1066-4f5e-11ef-b622-ef8c2e578b08'. Below the executor name, there is a list of metadata including environment label, workflow ID, address, provider, launcher, and worker initialization command. A table titled 'Used by Tasks:' lists the tasks associated with this executor, including 'initialize', 'split_data', 'process', and 'combine'.

Parsl Parsl Workflows File Provenance Documentation

Executor (fc9d1066-4f5e-11ef-b622-ef8c2e578b08)

- **Environment label:** local_htex
- **Workflow id:** [provenance_test.py](#)
- **Address:** None
- **Provider:** slurm
- **Launcher:** SrunLauncher
- **Worker init:** module load python3
- **Used by Tasks:**

Id	Name
0	initialize
1	split_data
2 3 4 5	process
6	combine

Dynamically Created Files

One issue we came across are dynamically created files

- A File which is created in an App and appended to the outputs
 - This often happens when you don't know how many files an App will produce
 - Unfortunately Parsl does not “know” about these files, they will not be transferred and cannot be used as inputs to another App
- Our solution is the Dynamic File List
 - It acts just like a list, but is also a Future
 - It updates the Dataflow Kernel about new files
 - Allows for Apps to rely on files that have no references at run time

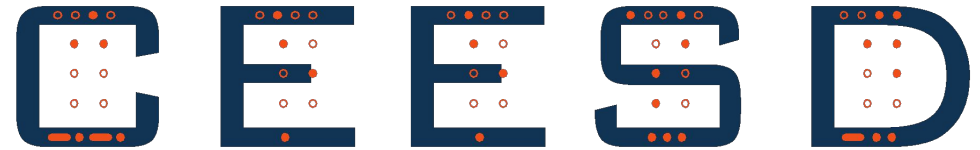


Dynamically Created Files

An example

```
@python_app
def produce(outputs=[]):
    def analyze(i):
        f =
File(f'file://path/to/file{i}.l
og}')
        with open(f.filepath,
'w') as out:
            # do some kind of
analysis
            return f
            count = int(random() * 10)
            fl =
File(f'file://path/to/master.lo
g')
            outputs.append(fl)
            with open(fl.filepath, 'w')
as log:
                log.write(f'Producing
{count} files\n')
                for i in range(count):
                    log.write(f"Running
analysis {i}\n")
            outputs.append(analyze(i))
```

```
@python_app
def produce(outputs=[]):
    def analyze(i):
        f =
File(f'file://path/to/file{i}.l
og}')
        with open(f.filepath,
'w') as out:
            # do some kind of
analysis
            return f
            count = int(random() * 10)
            fl =
File(f'file://path/to/master.lo
g')
            outputs.append(fl)
            with open(fl.filepath, 'w')
as log:
                log.write(f'Producing
{count} files\n')
                for i in range(count):
                    log.write(f"Running
analysis {i}\n")
            outputs.append(analyze(i))
```

Questions?

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number *DE-NA0003963*.

