ILLINOIS
NCSA | National Center for Supercomputing Applications

Building Coherence between Parsl and CWL

Yuhang Ren, Undergraduate SPIN Intern
Daniel S. Katz, Assistant Director for Scientific Software and Applications

Introduction

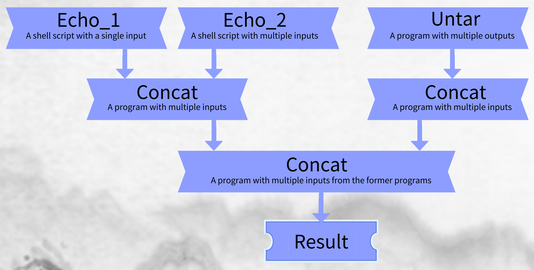
The central topic of this project is to build coherence between Parsl and CWL, two parallel programming solutions to the problem of running a series of application tasks (apps) with dependencies. To achieve such coherence, I studied how to define CWL apps based on defined apps in Parsl programs, as well as how to capture the data about a Parsl program after it has been run into a machine-written CWL workflow.

Motivation

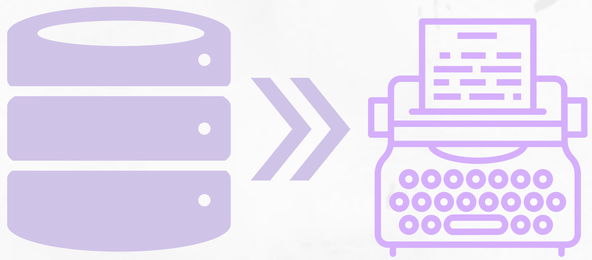
While Parsl provides substantial parallel programming support for Python, it does so in a different way from what is specified by Common Workflow Language (CWL), a specification common in the life sciences (and to a lesser extent, in other fields) for describing analysis workflows and tools in a way that makes them portable and scalable across a variety of software and hardware environments.

Solution

1. A set of restrictions on Parsl programming:
   a) Avoid using Python functions and environment variables in bash_apps.
   b) Limit the input and output data types of apps.
   c) Include the non-input arguments in the input keyword.
2. A program that reproduces a Parsl workflow into a CWL workflow that achieves identical purposes, which can be found at :
   https://github.com/Parsl/student-projects



task_id	task_depends	task_func_name	task_inputs	task_outputs
0		echo_1	[',/echo1.s...	['echo1.txt']
1		echo_2	[',/echo2.s...	['echo2.txt']
2		untar	['texts,tar']	['1.txt', '...
3	0, 1	concat	[echo1.txt,...	['NCSA.txt']
4	3, 3	concat	[1.txt, 2.t...]	['SPIN.txt']
5	2, 4	concat	[NCSA.txt, ...]	['Result.tx...]



COMMON WORKFLOW LANGUAGE

Parsl Workflow

While Python is a well - known programming language widely used due to its simplicity, ability to integrate with other languages, and its interpreted nature, it lacks ability to run programs in parallel. One means of allowing parallel programming in Python is Parsl, a Python library for programming and executing data-oriented workflows (dataflows) in parallel. Python programs that use Parsl allow selected Python functions and external applications (called apps) to be connected by shared input/output data objects into flexible parallel workflows.

```
@bash_app
def concat(inputs=[], outputs=[], stdout=stdout.txt, stderr=stderr.txt):
    command = '{exe} {file_0} {file_1} > {output}'.format(
        exe    = 'cat',
        file_0 = inputs[0],
        file_1 = inputs[1],
        output = outputs[0]
    )
    return command

future = concat(inputs=['Intelligence','Dilligence'], outputs=['NCSA'])
```

MonitoringHub

Parsl

When implemented appropriately, Parsl can provide a monitoring module, called MonitroingHub, that tracks each task executed and sotre the information of them into a database. Information in the database includes task IDs, task function names, inputs and outputs, as well as dependencies of each task. Such information serves as a clue that can be interpreted and used to reproduce the procedure of that program.

Translator

Running information stored in the database provided by MonitoringHub makes a translator program possible, which interprets the information and writes CWL scripts that performs identical tasks as the interpreted Parsl program. The translator stores task information according to templates of CWL CommandLineTools (the equivalence of bash_app in Parsl) and CWL workflows, after which it prints the CWL scripts into several files.

```
auto_workflow_1 = cwl_Workflow(inputs=[], outputs=[],steps=[])
interpret(Task, db, auto_workflow_1, clts)
render_cwl("auto_workflow.cwl",auto_workflow_1)
```

CWL Workflow

```
class: CommandLineTool
baseCommand: cat
stdout: $(inputs.input_2)
inputs:
    input_0: type: File
    input_1: type: File
    input_2: type: string
outputs:
    output_0: type: File
      outputBinding:
        glob: $(inputs.input_2)
```

```
input_0:
  class: File
  path: Intelligence
input_1:
  class: File
  path: Dilligence
input_2: NCSA
```

Above is a sample code snippet that can be generated by the translator. It follows the grammar of CWL, and can produce the same result as the Parsl workflow translated. Such translation provides coherence between Parsl and CWL in that Parsl programmers can program without worrying about rewriting the same program in CWL, which can be impractical and inefficient. The translator can serve as a bridge leading Parsl programmers to write CWL-friendly programs.

Sponsored by NCSA SPIN program and NSF award 1550588