b2luigi

bringing batch to luigi

Maximilian Welsch - mwelsch@uni-bonn.de

13.11.2019

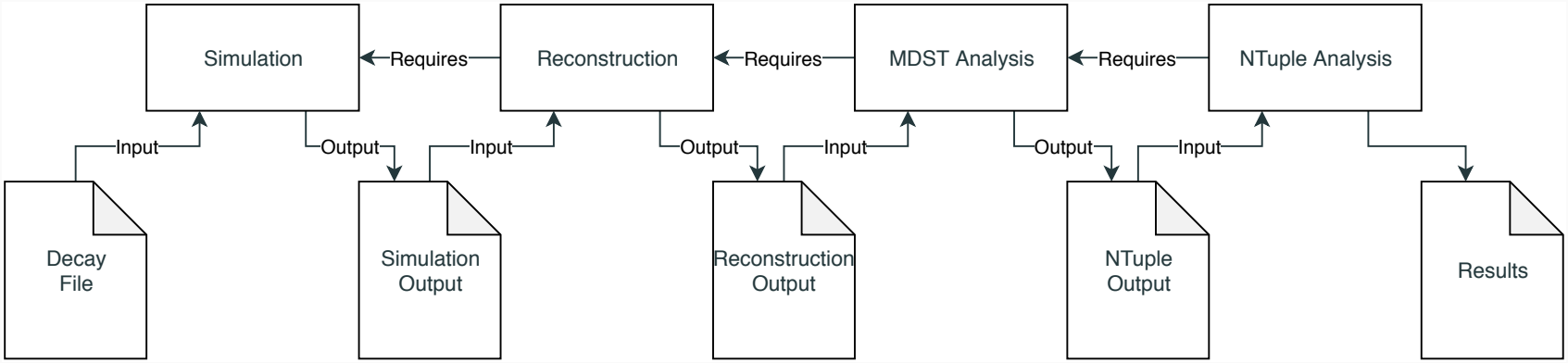
Physikalisches Institut
University of Bonn

Table of contents

Short Introduction to `luigi`

What is `luigi`

"`luigi` is a Python package that helps you build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization, handling failures, command line integration, and much more."



Avoid Manual Piplines

```
$ python simulation.py
$ python reconstruction.py
$ python analysis.py
$ python ntuple_analysis.py
```

- Pros:
  - Quick and hacky
- Cons:
  - Manually rerun the sequence of scripts in right order
  - Deal with input and output files and their dependencies

Building Workflows

See `luigi` documentation

- Workflows are built using the `Task`, `Target` and `Parameter` classes provided by luigi
- `Task`
  - Logical unit of work
  - Need to implement `requires`, `output` and `run` methods
  - Dependencies between task defined by their input and output
- `Target`
  - Corresponds (in our case) to a file on disk
  - Defines both output and input for tasks
- `Parameter`
  - Allows for some form of parametrization for tasks
  - e.g. number of events, event type, ...

Example - MyNumberTask

```python
import luigi
import random

class MyNumberTask(luigi.Task):
    some_parameter = luigi.Parameter()

    def output(self):
        return luigi.LocalTarget(
            f"output_file_{self.some_parameter}.txt"
            )

    def run(self):
        random_number = random.random()

        with self.output().open("w") as f:
            f.write(f"{random_number}\n")
```

Example - MyAverageTask

```python
class MyAverageTask(luigi.Task):
    def requires(self):
        for i in range(100):
            yield MyNumberTask(some_parameter=i)

    def output(self):
        return luigi.LocalTarget("average.txt")

    def run(self):
        # Build the mean
        summed_numbers = 0
        counter = 0

        for input_file in self.input():
            with input_file.open("r") as f:
                summed_numbers += float(f.read())
                counter += 1

        average = summed_numbers / counter

        with self.output().open("w") as f:
            f.write(f"{average}\n")
```

b2luigi

What is `b2luigi`

- Built on top of `luigi` for automatically scheduling tasks locally or on a batch system
- Extends the `luigi` user interface and has a build-in support for queue systems, e.g. LSF or HTCondor.

```python
import b2luigi

class MyTask(b2luigi.Task):
    def output(self):
        return b2luigi.LocalTarget("output_file.txt")

    def run(self):
        with self.output().open("w") as f:
            f.write("This is a test\n")


if __name__ == "__main__":
    b2luigi.process(MyTask(), batch=True)
```

Motivation for `b2luigi`

Why not use already created batch tasks[1]?

- Run many (many many!) batch jobs in parallel
- Existing large set of luigi tasks in project
- Be independent of batch system

Is that all?

- `b2luigi` provides helper functionalities that help you with large projects
- `b2luigi.Task` is a super-hero version of `luigi.Task` that helps with file management
- Automatic creation of result directories based on task parameters
- Developed for the Belle II experiment → `basf2` specific helpers

[1] `https://github.com/spotify/luigi/blob/master/luigi/contrib/sge.py`

Quick Start

```python
import b2luigi
import random

class MyNumberTask(b2luigi.Task):
    some_parameter = b2luigi.IntParameter()

    def output(self):
        yield self.add_to_output("out_file.txt")

    def run(self):
        random_number = random.random()

        with open(self.get_output_file_name("out_file.txt"), "w") as f:
            f.write(f"{random_number}\n")

if __name__ == "__main__":
    b2luigi.set_setting("result_path", "results")
    b2luigi.process(
        [MyNumberTask(some_parameter=i) for i in range(100)],
        workers=200,
        batch=True
    )
```

Batch Processing

- Choosing the the batch system
  - HTCondor (NAF) or LSF (KEKCC)
- Choosing the environment
  - LSF: not necessary, worker node uses same environment as submission node
  - HTCondor: necessary, just provide a simple shell script
- Filesystem
  - Default: result and script folder have to be accessible from worker node
  - If not, e.g. use file copy mechanisms specific to batch system
- Drawbacks
  - queue/batch setttings/etc. have to known beforehand
  - No automatic resubmission
  - Dynamic luigi dependencies via `yield` not implemented in batch mode

Links

- `https://github.com/spotify/luigi`
- `https://luigi.readthedocs.io/en/stable/`
- `https://github.com/nils-braun/b2luigi` (PRs welcome!)
- `https://b2luigi.readthedocs.io/en/stable/index.html`