Jug: Reproducible Research in Python

Luis Pedro Coelho

EMBL

5 July 2013
def preprocess(f):
    return ...

def compute(fs, param):
    return ...

def write_output(results):
    ...

intermediate = []
for i in glob('* .txt '):
    intermediate.append(processing(i))
results = []
for pvalue in [0.5, 1.0, 2.0, 4.0]
    results.append(compute(intermediate, pvalue))
write_output(results)
A Processing Pipeline in JUG

@TaskGenerator
def preprocess(f):
    return . . .

@TaskGenerator
def compute(fs, param):
    return . . .

@TaskGenerator
def write_output(results):
    . . .

intermediate = []
for i in glob('*.txt'):
    intermediate.append(processing(i))
results = []
for pvalue in [0.5, 1.0, 2.0, 4.0]
    results.append(compute(intermediate, pvalue))
write_output(results)
$ jug execute &
[1] 20332
$ jug execute &
[2] 20333
$
Jug Enhances Reproducibility

**Dark Side of Computational Analysis**

- “What was the parameter that generated this result? I think it was $\frac{1}{2}$, right? Had to be.”
- “Deleted the intermediate results, reran; now everything is different.”
- “We cannot reproduce the table in our own paper.”

**Advantages of Jug**

- With jug, changing parameters will trigger recomputation of all downstream results.
- `jug invalidate` handles all dependencies
- Unlike `make`, you can use any Python function
Finding Out More About Jug...

- Talk to me **in person**
- luis@luispedro.org
- [http://github.com/luispedro/jug](http://github.com/luispedro/jug) the code
- [http://jug.rtfd.org](http://jug.rtfd.org) read the fine documentation