Custom templates for streamlined DAG workflows

Christopher Cox

University of Wisconsin—Madison

May 21, 2015
Why I use HTCondor

- I am an aspiring Cognitive Neuroscientist.
Why I use HTCondor

- I am an aspiring Cognitive Neuroscientist.
- I think of the brain as a vast, complicated network that utilizes distributed representations.
Why I use HTCondor

- I am an aspiring Cognitive Neuroscientist.
- I think of the brain as a vast, complicated network that utilizes distributed representations.
- Studying the brain bases of mental representations is computationally intensive.
Why I use HTCondor

- Equally vast are the parameter spaces that must be explored.
Why I use HTCondor

- Equally vast are the parameter spaces that must be explored.
- Parameter selection and model performance must be cross validated.
Why I use HTCondor

- Equally vast are the parameter spaces that must be explored.
- Parameter selection and model performance must be cross validated.
- Null (hypothesis) distributions need to be estimated from many random permutations.
Why I use HTCondor

- Equally vast are the parameter spaces that must be explored.
- Parameter selection and model performance must be cross validated.
- Null (hypothesis) distributions need to be estimated from many random permutations.

Analyses are complex, and require thousands of different iterations.
Why I use HTCondor

- Equally vast are the parameter spaces that must be explored.
- Parameter selection and model performance must be cross validated.
- Null (hypothesis) distributions need to be estimated from many random permutations.

Analyses are complex, and require thousands of different iterations.
My typical workflow

Sweep  Summarize  Final Fit
My typical DAG

workflow.dag

SPLICE A A/sweep.dag
SPLICE B B/summarize.dag
SPLICE C C/finalfit.dag
PARENT A CHILD B
PARENT B CHILD C
It all looks so simple . . .

- A B C, easy as . . . ♫
It all looks so simple . . .

- A B C, easy as . . .
- There are many little files to generate keep track of.
It all looks so simple . . .

- A B C, easy as . . .♫
- There are many little files to generate keep track of.
  - Each job has a submit file, and may be wrapped in a sub-DAG
It all looks so simple . . .

- A B C, easy as . . .
- There are many little files to generate keep track of.
  - Each job has a submit file, and may be wrapped in a sub-DAG
  - Jobs (or sub-DAGs) are enumerated in the DAGs references in workflow.dag.
It all looks so simple . . .

- A B C, easy as . . .🎵
- There are many little files to generate keep track of.

  - Each job has a submit file, and may be wrapped in a sub-DAG
  - Jobs (or sub-DAGs) are enumerated in the DAGs references in workflow.dag.
  - Everything is parameterized in many different particular ways.
It all looks so simple . . .

- A B C, easy as . . . 🎵
- There are many little files to generate keep track of.
  - Each job has a submit file, and may be wrapped in a sub-DAG
  - Jobs (or sub-DAGs) are enumerated in the DAGs references in workflow.dag.
  - Everything is parameterized in many different particular ways.
- This workflow is likely run many times, with different variations.
It all looks so simple . . .

- A B C, easy as . . .￼
- There are many little files to generate keep track of.
  
  - Each job has a submit file, and may be wrapped in a sub-DAG
  - Jobs (or sub-DAGs) are enumerated in the DAGs references in workflow.dag.
  - Everything is parameterized in many different particular ways.

- This workflow is likely run many times, with different variations.
It all looks so simple . . .

- A B C, easy as . . .🎶
- There are many little files to generate keep track of.
  - Each job has a submit file, and may be wrapped in a sub-DAG
  - Jobs (or sub-DAGs) are enumerated in the DAGs references in workflow.dag.
  - Everything is parameterized in many different particular ways.
- This workflow is likely run many times, with different variations.
My pitch

Much of this can be *easily automated* in a way that is *easy to maintain* and *intuitive to work with*. 
Perl and Text::Template

Requirements

1. *Basic* working knowledge of Perl. (I’m proof that you don’t need to know much.)
2. Getting comfortable with JSON (and/or YAML).
3. The Text::Template module for Perl.
Overview

- Each file that you need to generate many times will need a template.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl "hash").
  4. Pass that hash into the template to fill it in.
  5. Write the filled template to a file.

This is the same way some dynamic web pages are constructed.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl “hash”).
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl “hash”).
  4. Pass that hash into the template to fill it in.

This is the same way some dynamic web pages are constructed.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl “hash”).
  4. Pass that hash into the template to fill it in.
  5. Write the filled template to a file.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl "hash").
  4. Pass that hash into the template to fill it in.
  5. Write the filled template to a file.
Overview

- Each file that you need to generate many times will need a template.
- Each template will be associated with a short Perl program that will (at minimum):
  1. Read the template, and prepare it for filling.
  2. Read a parameter file (either JSON or YAML formatted).
  3. Represent those parameters as a key-value based data structure (a Perl “hash”).
  4. Pass that hash into the template to fill it in.
  5. Write the filled template to a file.

This is the same way some dynamic web pages are constructed.
Example (YAML) parameter file

# If your jobs are less than 4 hours long, "flock" them ...
FLOCK: "false"

# If your jobs are less than ~2 hours long, "glide" them ...
GLIDE: "false"

# Specify in KB, MB, or GB. No space between numbers and letters.
request_memory: "4GB"
request_disk: "10GB"
Example template

```plaintext
# If your jobs are less than 4 hours long, "flock" them ...  
+WantFlocking = <$FLOCK$>

# If your jobs are less than ~2 hours long, "glide" them ...  
+WantGlidein = <$GLIDE$>

# Tell Condor how many CPUs (cores), how much memory (MB) and how much disk space (KB) each job will need:  
request_cpus = 1
request_memory = <$MEM_MB$>
request_disk = <$DISK_KB$>
```
Example header for Perl program

```
#!/usr/bin/env perl
use strict;
use warnings;
use YAML::XS; # or JSON::Parse
use Text::Template;
```

Optional, but often useful:
```
use String::Scanf;
use Data::Dumper;
use Path::Tiny qw(path);
```

Data::Dumper lets you print hashes to the screen, and Path::Tiny contains a function for composing absolute paths.

---

1 https://github.com/crcox/condortools
Example header for Perl program

```perl
#!/usr/bin/env perl
use strict;
use warnings;
use YAML::XS; # or JSON::Parse
use Text::Template;
```

Optional, but often useful:

```perl
use String::Scanf;
use Data::Dumper;
use Path::Tiny qw(path);
```

Data::Dumper lets you print hashes to the screen, and Path::Tiny contains a function for composing absolute paths.

---

1https://github.com/crcox/condortools
expandStub.py and expandStub_yaml.py

Ok, but what about all the parameter files you need to write?

2https://github.com/crcox/condortools
Ok, but what about all the parameter files you need to write? I have written a couple Python tools that can help with this.²

²https://github.com/crcox/condortools
expandStub_yaml.py demo

If I have a file called stub.yaml that contains:

```yaml
lambda: [1, 2, 3, 4]
targets: "animals"
Gtype: "L1L2"
ExpandFields:
  - lambda
```

expandStub_yaml.py stub.yaml will produce a file named master.yaml that contains:

```yaml
--- {Gtype: L1L2, lambda: 1, targets: animals}
--- {Gtype: L1L2, lambda: 2, targets: animals}
--- {Gtype: L1L2, lambda: 3, targets: animals}
--- {Gtype: L1L2, lambda: 4, targets: animals}
```
Take home

- These basic tools can be combined to automate the creation of files that every project needs.
Take home

▶ These basic tools can be combined to automate the creation of files that every project needs.
▶ A stub file can be expanded into a “master” file where all combinations of flagged parameters will be expanded into an easily looped form.
Take home

- These basic tools can be combined to automate the creation of files that every project needs.
- A stub file can be expanded into a “master” file where all combinations of flagged parameters will be expanded into an easily looped form.
- Such a loop can be implemented within a Perl program to generate files based on a template.
Thank you

https://github.com/crcox/condortools