Submitting Multiple Jobs With HTCondor

Christina Koch

HTCondor Week 2020
Why multiple jobs?
Why multiple jobs?

Mei Monte Carlo

Needs to run many random simulations to model particles in a detector

Image credit: The Carpentries Instructor Training
## Why multiple jobs?

<table>
<thead>
<tr>
<th>Mei Monte Carlo</th>
<th>Tamara Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs to run many random simulations to model particles in a detector</td>
<td>Testing different design parameters for designing clinical trials.</td>
</tr>
</tbody>
</table>

Image credit: [The Carpentries Instructor Training](https://www.thecarpentries.org/instructor-training)
## Why multiple jobs?

<table>
<thead>
<tr>
<th>Mei Monte Carlo</th>
<th>Tamara Trials</th>
<th>Ben Bioinformatics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs to run many random simulations to model particles in a detector</td>
<td>Testing different design parameters for designing clinical trials.</td>
<td>Applying a quality control / processing pipeline to 20 RNA samples.</td>
</tr>
</tbody>
</table>

Image credit: [The Carpentries Instructor Training](#)
Multiple job goals

<table>
<thead>
<tr>
<th>Mei Monte Carlo</th>
<th>Tamara Trials</th>
<th>Ben Bioinformatics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs to run many random simulations to model particles in a detector.</td>
<td>Parameters for designing clinical trials.</td>
<td>Applying a quality control / processing pipeline to 20 RNA samples.</td>
</tr>
</tbody>
</table>

**TO AVOID:**
- starting each job manually
- creating separate submit files for each job

Image credit: The Carpentries Instructor Training
Many jobs, one submit file

HTCondor has several built-in ways to submit multiple independent jobs from one submit file
Let’s review: one job

```bash
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in

log    = job.log
output = job.stdout
error  = job.stderr

queue
```

This is the command we want HTCondor to run.
Let’s review: one job

```plaintext
executable = analyze.sh
arguments = file.in file.out
transfer_input_files = file.in
log = job.log
output = job.stdout
error = job.stderr
queue
```

These are the files we need for the job to run.
Let’s review: one job

executable = analyze.sh
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.stdout
error = job.stderr

queue

These files track information about the job.
Example 1: Many jobs with numbered files

Now suppose we have many input files and we want to run one job per input file.
List of numerical input values

We want to capture this set of inputs using a list of integers.

file.0.in
file.1.in
file.2.in
file.3.in
file.4.in
Provide a list of integer values with `queue N`

executable = analyze.sh
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.stdout
error = job.stderr

queue 5

This queue statement will generate a list of integers, 0 - 4
Which job components vary?

```bash
executable = analyze.sh
arguments  = file.in file.out
transfer_input_files = file.in
log        = job.log
output     = job.stdout
error      = job.stderr
queue 5
```

The arguments for our command and the input files would be different for each job.
Which job components vary?

```bash
executable = analyze.sh
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.stdout
error = job.stderr

queue 5
```

We might also want to differentiate these job files.
Use $(\text{ProcID})$ as the variable

```plaintext
executable = analyze.sh
arguments = file.$(\text{ProcID}).in file.$(\text{ProcID}).out
transfer_input_files = file$(\text{ProcID}).in

log = job.$(\text{ProcID}).log
output = job.$(\text{ProcID}).stdout
error = job.$(\text{ProcID}).stderr

queue 5
```

The default variable representing the changing numbers in our list is $(\text{ProcID})$
Example 2: Many jobs with named files

• Program execution
  
  $ \texttt{compare\_states state.wi.dat out.state.wi.dat}$

• Files needed
  
  • compare\_states, state.wi.dat, country.us.dat

  executable = compare\_states
  arguments = state.wi.dat out.state.wi.dat

  transfer\_input\_files = state.wi.dat, country.us.dat

  queue
List of named input values

• Suppose we have data for several states: state.wi.dat, state.mn.dat, state.il.dat, etc.
• We want to run one job per file.

```python
executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue
```
Provide a list of values with `queue from`

- We want to use “queue” to provide this list of input files.
- One option is to create another file with the list and use the `queue ... from` syntax.

```plaintext
executable = compare_states
arguments = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue from state_list.txt
```
Which job components vary?

• Now, what parts of our job template (the top half of the submit file) vary, depending on the input?
• We want to vary the job’s **arguments** and one **input file**.

```bash
executable = compare_states
arguments  = state.wi.dat out.state.wi.dat
transfer_input_files = state.wi.dat, country.us.dat
queue state from state_list.txt
```
Use a custom variable

• Replace all our varying components in the submit file with a variable.

```plaintext
executable = compare_states
arguments = $(state) out.$(state)

transfer_input_files = $(state), country.us.dat

queue state from state_list.txt
```
Use multiple variables with `queue from`

- The queue from syntax can also support multiple values per job.
- Suppose our command was like this:

  ```bash
  executable = compare_states
  arguments = -i $(state) -y $(year)
  transfer_input_files = $(state), country.us.dat
  queue state,year from state_list.txt
  ```
## Variable and `queue` options

<table>
<thead>
<tr>
<th>Syntax</th>
<th>List of Values</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queue N</code></td>
<td>Integers: 0 through N-1</td>
<td>$(ProcId)$</td>
</tr>
<tr>
<td><code>queue Var matching pattern*</code></td>
<td>List of values that match the wildcard pattern.</td>
<td>$(Var)$</td>
</tr>
<tr>
<td><code>queue Var in (item1 item2 ...)</code></td>
<td>List of values within parentheses.</td>
<td>If no variable name is provided, default is $(Item)$</td>
</tr>
<tr>
<td><code>queue Var from list.txt</code></td>
<td>List of values from <code>list.txt</code>, where each value is on its own line.</td>
<td></td>
</tr>
</tbody>
</table>
Other options: `queue N`

- Can I start from 1 instead of 0?
  - Yes! These two lines increment the `$(ProcId)` variable
    ```
    tempProc = $(ProcId) + 1
    newProc = $INT(tempProc)
    ```

  - You would use the second variable name `$(newProc)` in your submit file

- Can I create a certain number of digits (i.e. 000, 001 instead of 0,1)?
  - Yes, this syntax will make `$(ProcId)` have a certain number of digits
    ```
    $INT(ProcId, %03)
    ```
Other options: `queue in/from/matching`

- You can run multiple jobs per list item, using $(Step) as the index:

  ```
  executable = analyze.sh
  arguments = -input $(infile) -index $(Step)
  queue 10 infile matching *.dat
  ```

- `queue matching` has options to select only files or directories

  ```
  queue inp matching files *.dat
  queue inp matching dirs job*
  ```
Case Study 1

- What varies?
  - Not much – just needs an index to keep simulation results separate.

- Use queue N
  - Simple, built-in
  - No need for specific input values

Mei Monte Carlo

Needs to run many random simulations to model particles in a detector
Case Study 2

• What varies?
  • Five parameter combinations per job
  • Parameters are given as arguments to the executable

• Use `queue from`
  • `queue from` can accommodate multiple values per job
  • Easy to re-run combinations that fail by using subset of original list
Case Study 3

- What varies?
  - Each job analyzes one sample; each sample consists of two fastq files in a folder with a standard prefix.

- Use queue matching
  - Folders have a standard prefix, input files have standard suffix, easy to pattern match

- Good alternative: queue from
  - Provide list of folder names/file prefixes, construct paths in the submit file.

- Want output files to return to the same folder (stay tuned...)
## Queue options, pros and cons

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue $N$</td>
<td>Simple, good for multiple jobs that only require a numerical index.</td>
</tr>
<tr>
<td>queue matching pattern*</td>
<td>Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories. Requires good naming conventions.</td>
</tr>
<tr>
<td>queue in list</td>
<td>Supports multiple variables, all information contained in a single file, reproducible. Harder to automate submit file creation.</td>
</tr>
<tr>
<td>queue from file</td>
<td>Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible. Additional file needed.</td>
</tr>
</tbody>
</table>
Many jobs means many files.
Directories are your friends

```bash
executable = analyze.sh
transfer_input_files = input/file$(ProcID).in, shared/
log = logs/job.$(ProcID).log
output = output/job.$(ProcID).stdout
error = error/job.$(ProcID).stderr
queue 5
```

```
submit_dir/
  jobs.submit
  analyze.sh
shared/
  script1.sh
  reference.dat
input/
  file0.in
...
logs/
  job.0.log
...
output/
  job.0.stdout
...
error/
  job.0.stderr
...```
Job-specific directories with `initialdir`

```sh
executable = analyze.sh
transfer_input_files = file.in
initialdir = job$(ProcId)

output = job.stdout
everror = job.stderr

queue 5
```

submit_dir/
jobs.submit
analyze.sh
job0/
   file.in
   job.stdout
   job.stderr
job1/
   file.in
   job.stdout
   job.stderr
job2/
   ...
```
Use variables, move output files

infile = file$(ProcID).in
outfile = file$(ProcID).out
executable = analyze.sh
arguments = $(infile) $(outfile)

transfer_input_files = input/$(infile)
transfer_output_files = $(outfile)
transfer_output_remaps = "$(outfile)=output/$(outfile)"

queue 5
Resources

• Example jobs and submit files:
  • https://github.com/CHTC/example-multiple-jobs

• condor_submit documentation:
  • Search for “queue”

• HTCondor user tutorial
  • https://agenda.hep.wisc.edu/event/1325/session/0/contribution/19/material/slides/0.pdf

• Advanced submit talk
  • https://agenda.hep.wisc.edu/event/1325/session/3/contribution/40/material/slides/0.pptx
Questions?