

Submitting Multiple Jobs

European HTCondor Workshop
Oct 2022

Todd Tannenbaum
Center for High Throughput Computing
Department of Computer Sciences
University of Wisconsin-Madison

Why multiple jobs

› User perspective


- Parameter sweep is a natural HTC model
- No need to create a separate submit file for each job
- Single "handle" to query, remove, hold, edit many jobs
- Aggregates (e.g. # idle, # completed, # held) across many jobs

› Admin perspective

- **Much** less resources consumed on the AP
 - Both schedd memory and processing load

Overview

HTCondor has several built-in ways to create multiple independent jobs

- › condor_submit Queue statement in job submit file
 - Queue N
 - Queue Matching
 - Queue From
 - Queue In
- › Late Materialization
- › Job Sets 

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
```

This is the command we want HTCondor 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 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.



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?

```
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?

```
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 `$ (ProcID)` as the variable

```
executable = analyze.sh
arguments  = file.$ (ProcID) .in file.$ (ProcID) .out
transfer_input_files = file$ (ProcID) .in

log        = job.$ (ProcID) .log
output     = job.$ (ProcID) .stdout
error      = job.$ (ProcID) .stderr

queue 5
```

The default variable representing the changing numbers in our list is `$ (ProcID)`

Example 2: Many jobs with named files

› Program execution

```
$ 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.

```
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`.

```
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
```

```
state.wi.dat
state.mn.dat
state.il.dat
state.ia.dat
state.mi.dat
```

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**.

```
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.

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

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

```
queue state from state_list.txt
```



```
state.wi.dat  
state.mn.dat  
state.il.dat  
state.ia.dat  
state.mi.dat
```

Use multiple variables with queue from

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


```
$ compare_states -i [input file] -y [year]
```

```
executable = compare_states  
arguments  = -i $(state) -y $(year)
```

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

```
queue state,year from state_list.txt
```

```
state.wi.dat,2010  
state.wi.dat,2015  
state.mn.dat,2010  
state.mn.dat,2015
```



Variable and queue options

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

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*
```

Late Materialization

What if you want to submit thousands or even millions of jobs?

- › Job submit file options:
 - **max_materialize = <limit>** : specifies an overall limit on the number of jobs that can be materialized in the *condor_schedd* at any one time
 - **max_idle = <limit>** : specifies the maximum number of non-running jobs that should be materialized in the *condor_schedd* at any one time.
- › Get around admin AP limits like MAX_JOBS_PER_OWNER or MAX_JOBS_PER_SUBMISSION
- › Returns a handle to the "job factory" ID
- › Details: Manual / Users Manual / Submitting a Job / Submitting Lots of Jobs

Queue command options, pros and cons

<code>queue N</code>	Simple, good for multiple jobs that only require a numerical index.
<code>queue matching pattern*</code>	Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories Requires good naming conventions.
<code>queue in (list)</code>	Supports multiple variables, all information contained in a single file, reproducible Harder to automate submit file creation
<code>queue from file</code>	Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible Additional file needed

Queue command CONS

- › Creates a "job cluster" ID, which really just captures all jobs created in one submission.
- › Problems:
 - What if you want to group jobs across multiple job submissions?
 - Add jobs later on
 - DAGMan
 - Want to group different submit files into one set ?
 - What if you want to label your group of jobs?
 - Want attributes/policies in common across many jobs?
- › Answer: Job Sets!

Job Sets

- › Enable feature in AP with config knob "USE_JOBSETS=True"
- › Use condor_submit with "**jobset = XXX**" (boo!) or new tool "**htcondor jobset <verb>**" (yay!)
 - htcondor jobset create
 - htcondor jobset list
 - htcondor jobset destroy
- › Job Set consists of
 - A name (assigned by the user)
 - An iterator
 - At least one job
 - A job set ID (assigned by the AP)

htcondor jobset create "myset.sub"

```
name = MyJobSet
```

```
iterator=table x,y,z params.txt
```

```
job in=x,foo=y,bar=z myjob.sub
```

```
job in=x,p1=y,p2=z myjob2.sub
```

```
name = MyJobSet
```

```
iterator = table x,y,z {
```

```
    input_A.txt,0,0
```

```
    input_B.txt,0,1
```

```
    input_C.txt,1,0
```

```
    input_D.txt,1,1
```

```
}
```

```
job input=x,param1=y,delta=z {
```

```
    executable = a.out
```

```
    arguments = $(input) $(param1) $(delta)
```

```
    transfer_input_files = $(input)
```

```
}
```

Thank You!



Follow us on Twitter!
<https://twitter.com/HTCondor>



This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project.

PATh PARTNERSHIP to ADVANCE
THROUGHPUT
COMPUTING